Sammendrag

NTNU SOC ønsker en utredning for hvor godt en konteinerbasert versjon av stordataanalyseverktøyet Apache Spark, som kjører i konteinerorkestreringsplattformen Openshift Kubernetes Distribution (OKD), kan bidra til avansert sikkerhetsanalyse hos NTNU. Rapporten tar for seg installasjon og konfigurasjon av OKD og hvordan de ulike støttetjenestene OKD er avhengig av settes opp. Det forklares hvordan denne infrastrukturen kan benyttes for stordataanalyse og tar for seg batch prosessering av data fra Hadoop Distributed File System.

Abstract

NTNU wants to conduct an evaluation if running containerized Apache Spark inside the container orchestration platform Openshift Kubernetes Distribution(OKD) can contribute to advanced security analysis at NTNU. The report covers the installation and configuration process of OKD and all of the required services OKD is dependent on. Further an explanation of how this infrastructure can be used for batch processing of data from Hadoop Distributed File System used in big data analysis is presented.

Forord

Forfatterne av denne bacheloroppgaven, Alf Pettersen, Anders Moen og Simon Røe, ønsker å takke vår veileder Christian Johansen for gode råd, veiledning og hjelp til å fullføre oppgaven.

Vil også takke oppdragsgiver for oppgaven representert av Christoffer Vargtass Hallstensen.

En siste takk går til Lars Erik Pedersen for hjelp med feilsøking av problemer vi sannsynligvis ikke ville funnet ut av selv innenfor den tidsrammen vi hadde.

Innhold

Figurer

Tabeller

Kodelister

Ordliste

arbeiderpod En node i Apache Spark der arbeid utføres. 21
arbeidsnode En node i OKD der arbeid utføres.. 26

chroot Endrer root folderen til en gitt prosess.. 12

driverpod Administrativ node i Apache Spark. 19

Helm Helm er Kubernetes sin package manager. 18
Helm chart En samling filer brukt av Helm for å utrullere en tjeneste på Kubernetes. 18

IaaS Infrastructure as a service er en skytjenste som tilbyr virutaliserte ressurser.. 12
image Et docker image er en mal for å skrive konfigurasjonsfiler som bygger konteinere. 16

kontrollplannode Administrativ node i OKD. 25

SOC Security operatons center. 1, 2

Kapittel 1

Innledning

Med den økende digitaliseringen av samfunnet, er det en stigende trend der bedrifter utsettes for digitale angrep. Høyere utdanningsinstitusjoner, som NTNU, må være forberedt på digitale angrep og er avhengige av en sikkerhetsavdeling for å oppdage, hindre og håndtere hendelser. Security operation center (SOC) er en funksjon som tilbyr sikkerhetstjenester som hendelseshåndtering, monitorering av nettverkstrafikk, og deteksjon av sikkerhetshendelser for, eller i en bedrift. Senteret håndterer store datamengder for å oppdage sikkerhetshendelser.

Ved å benytte en kraftig analysemotor kan man effektivt bearbeide store mengder data som kommer inn til SOC. En anerkjent metode for prossesering av data er Apache Spark som blir brukt av selskaper som Amazon, eBay og TripAdvisor[1]. Da Apache Spark sin eksperimentelle funksjonalitet for å kjøre i et Kubernetes miljø ble markert som produksjonsklar i Mars 2021, kan dette potensielt være en god løsning for å analysere data i konteinerbaserte kluster. Ved å kombinere analysemotoren med en orkestreringsplattform, tilrettelegger dette for analyse av stordata med en dynamisk tilnærming, der man kun bruker ressurser for analyse ved behov, og frigjør ressurser til andre programmer resten av tiden.

1.1 Bakgrunn

NTNU SOC er en del av Seksjon for Digital sikkerhet og har som oppgaver å detektere hendelser, utføre sikkerhetsanalyse og gjennomføre hendelseshåndtering hos NTNU[2].
I denne sammenhengen samles det inn store mengder data som inneholder informasjon som kan være nyttig i NTNU SOC sitt sikkerhetsarbeid. Dataene kan analyseres for å videreutvikle rutiner for deteksjon og hendelseshåndtering for å utbedre sikkerheten til organisasjonen. For å analysere denne informasjonen ønsker NTNU SOC å benytte en kombinasjon av Openshift Kubernetes Distribution (OKD), som er et program for administrering av konteinere, og

Apache Spark, et program for dataanalyse. NTNU SOC ønsker en redegjørelse for om det er en gunstig løsning å kombinere disse to teknologiene for deres brukstilfelle, og hvordan dette gjøres.

1.2 Problemområde

For å kunne oppdage og forbedre bedriftens evne til å motstå angrep samles og analyseres store mengder nettverkstrafikk og loggdata. Denne type data er ofte tidssensitiv, hvor tilgang til et system som enkelt kan skaleres opp eller ned basert på arbeidsmengde er essensielt for sikkerhetsarbeid.
For å kunne utføre sikkerhetsarbeid benyttes mange ulike verktøy. Flere av disse utvikles og vedlikeholes av frivillige personer uten garanti for kompatibilitet med systemene brukt av NTNU SOC. For å bruke disse vektøyene må de lastes ned på hver enkelt instans, og sørge for at systemet støtter de ulike avhengighetene påkrevd. Dette er nå en manuell prosess som ikke er skalerbar for å dekke behovet for funksjonaliteten de ulike verktøyene tilbyr.

1.3 Oppgavebeskrivelse

Oppgaven er produsert og utgitt av NTNU SOC basert på ønske om en utredning om disse systemene kan tas i bruk i sin infrastruktur.
I samarbeid med oppdragsgiver ble oppgaven presisert til å omfatte følgende punkter.

- Dokumentasjon på installasjon og oppsett av OKD kluster.
- Konfigurere Apache Spark til å kjøre i konteinere.
- System- og bruker-dokumentasjon.
- Forslag til arkitektur, skalering og lagring.
- Ytelsestest av Apache Spark på OKD.

Hovedoppgaven i prosjektet blir å få Apache Spark til å kjøre i en konteiner på et OKD kluster. Det skal dokumenteres hvordan det settes opp, hvilke ressurser det krever, og hvordan systemet kan benyttes. Det er også ønsket å vite mengder ressurser som kreves for å kjøre analyser.

1.4 Rammer og avgrensninger

1.4.1 Kravspesifikke rammer

Etter møter med oppdragsgiver ble det spesifisert at gruppen skal utvikle og teste løsningen i NTNU sin skyløsning SkyHigh. Dette på grunn av gruppens erfaring med infrastrukturen fra tidligere fag, mengden ressurser tilgjengelig på plattformen, samt mulighet for å jobbe med infrastrukturen uten å være tilstede fysisk. På grunn av koronaviruset var også dette alternativet mest hensiktsmessig i henhold til smittevern. Oppdragsgiver ønsker også å kunne

bruke Apache Spark med Spark applikasjoner skrevet i Python da dette er et språk de fleste har kjennskap til.
Dokumentasjon og konfigurasjonsfiler vil bli lagret i GIT og består av den praktiske delen av konfigurasjonen.

1.4.2 Avgrensninger

Denne rapporten tar for seg prosessene og infrastrukturen som blir benyttet for å lage et system med det formål å utføre regneoperasjoner i Apache Spark. Hvordan selve arbeidsoppgaven i Apache Spark blir konstruert, optimalisert, og hvordan den fungerer, er utenfor omfanget dekt i denne rapporten. Da det å lage Apache Spark jobber er utenfor omfanget av denne rapporten, vil testing bli gjennomført med eksempeloppgaver ut i fra den offisielle dokumentasjonen. Rapporten tar for seg de to infrastrukturene kalt produksjon, og test og utvikling. Siden test og utviklingsinfrastrukutren ikke er relevant for miljøet til NTNU SOC, vil ikke oppsett av denne infrastrukturen bli dokumentert.

1.5 Formål

Formålet med oppgaven er å undersøke hvilken muligheter en kombinasjon av Apache Spak og OKD kan tilby NTNU SOC. Resultatet framlagt i rapporten, skal være en veiledning for beslutningen NTNU SOC tar om å videreutvikle denne løsningen eller undersøke andre metoder for å løse problemstillingen. Resultatet vil legges ut som åpen kildekode og kan være til nytte for andre som søker videre informasjon om hvordan man setter opp en lignenende løsning.

1.6 Målgruppe

Denne oppgaven er gitt for å løse en utfording hos oppdragsgiver. Dokumentet struktureres og skrives for andre som har interesse av prosjektet. Andre målgrupper for denne rapporten kan være enkeltpersoner med en teknisk bakgrunn eller bedrifter som står ovenfor en lik problemstilling.

1.7 Om rapporten

Denne rapporten er skrevet på Norsk Bokmål. Det er også brukt engelske ord og uttrykk der det ikke finnes en god norsk oversettelse siden det er mer forståelig, samt enklere å bruke i videre søk etter informasjon. Rapporten benytter seg av lenker til andre kapitler og kilder som gjør leseropplevelsen bedre ved å benytte en digital utgave.

I oppgaven brukes to forskjellige infrastrukturer. Den første infrastrukturen skal ende opp hos oppdragsgiver og er referert til som *produksjon* i oppgaven. Den andre infrastrukturen er den gruppen tester og utvikler systemet på, og

er kalt *testing og utvikling*. De fleste av seksjonene i rapporten er relevante for begge infrastrukturene, men rapporten går også i gjennom deler som er relevant for enten produksjonsinfrastrukturen eller testing og utviklingsinfrastrukturen. For detaljert informasjon om hvilke kapitler som er relevant for hvilken infrastruktur se Rapportens struktur 1.7.1

I rapporten brukes terminologien *OKD* og *Kubernetes* om hverandre. Kubernetes er en stor del av OKD og hvilken terminologi som blir brukt i rapporten er avhengig av om konseptet som forklares er mest relatert til selve Kubernetes eller OKD.

1.7.1 Rapportens struktur

Rapporten er inndelt i syv kapitler. For å gi en oversikt er hvert enkelt kapittel kort beskrevet under.

1. Innledning
Introduksjon til oppgaven som presenterer generelle opplysninger som hva oppgaven går ut på, hva den forsøker å løse og hvilke avgrensingner som er satt.

2. Metode og kravspesifikasjon
Her blir metode gjennomgått og det er tatt for seg hvilke krav som stilles til løsningen som skal kjøres i produksjonsinfrastrukturen og inkluderer scenario for hvordan systemet er tiltenkt å brukes.

3. Teoritisk bakgrunn
Introduksjon til aktuelle teknologier og konsepter benyttet i oppgaven.

4. Designvalg og konfigurasjon av Apache Spark på OKD
Detaljert gjennomgang av strukturen til systemet, hvordan det settes opp og brukes. Kapitellet er delt opp i fem delkapitler. Delkapittel en handler om Apache Spark og er relevant for begge infrastrukturene og er uavhengig av hvilken infrastruktur som kjører under. Delkapittel to handler om Hadoop Distributed File System (HDFS), hvor relevant HDFS er for produksjonsinfrastrukturen og hvordan HDFS er satt satt opp i testing og utviklingsinfrastrukturen. Delkapittel tre og fire handler om ulike installasjonsmetoder og konfigurasjon som kan bli benyttet for å installere OKD for produksjonsinfrastrukturen.

5. Sikkerhet
Beskriver de ulike sikkerhetsmekanismene som de ulike teknologiene benytter, og hvordan de brukes.

6. Utprøving av dynamisk allokering på test og utviklingsinfrastruktur
Gjennomgang av hvordan systemet blir testet samt fremlegging av resultatene. Inneholder en beskrivelse av testing og utviklingsinfrastrukturen og test av dynamisk skalering.

7. Avslutning
Diskusjon og oppsummering av resultater samt fremlegging av en konklusjon om denne løsningen kan anbefales ut i fra satte krav.

Vedlegg
Inneholder ekstra materiale som ikke passer inn i rapporten men tilhører selve oppgaven. Her er blant annet den tekniske gjennomføringen av oppgaven beskrevet i vedlegg E, readme.md filen fra GIT som er en kolleksjon av kommandoer og steg for steg fremgangsmetoder som viser i praksis hvordan elementer skrevet om i teksten utføres, og konfigurasjonseksempler som er benyttet i oppsettet.

Kapittel 2

Metode og kravspesifikasjon

2.1 Prosjektgjennomføring

2.1.1 Arbeidsmetode

Hvert gruppemedlem er nødt til å ta ansvar for at deloppgaver blir gjennomført på en god måte, samt sørge for kontinuerlig fremdrift i prosjektet.
I denne sammenheng benyttes en utviklingsmetodologi for å ha klare retningslinjer som skal følges igjennom prosjektperioden. Gruppen har lite erfaring med flere av teknologiene i oppgaven, og kommer til å lære om teknologiene og konseptente nødvendig for å løse oppgaven underveis i prosjektperioden. Dette er hovedgrunnlaget for at den agile metoden Scrum med noen elementer fra Kanban er valgt. Alle gruppemedlemene er kjent med Scrum fra tidligere prosjektarbeid og har god forståelse for metodikken. Scrum gir ikke fleksibiliteten som er ønsket av gruppen for å kunne omprioritere oppgaver underveis i en sprint, derfor vil ikke alle prinsippene i Scrum bli inkludert i utviklingsmodellen. Oppgaven er delt inn i fire hovedfaser, forprosjekt, utvikling og implementering, testing og utredning i tillegg til ferdigstilling av rapport. Disse er igjen inndelt i mindre oppgaver som føres fortløpende inn i en kanban-tavle.

2.1.2 Roller

Oppgaven er gitt av NTNU SOC hvor Christoffer Vargtass Hallstensen, gruppeleder for SOC, er kontaktperson. Veileder under prosjektet er Christian Johansen, Førsteamanuensis på Institutt for informasjonssikkerhet og kommunikasjonsteknologi ved NTNU.

Gruppemedlemmer

Gruppen består av Alf Pettersen, Anders Wormdal Moen og Simon Rødsbakken Røe, som studerer IT-drift og Informasjonssikkerhet ved NTNU.

2.1.3 Innhenting av informasjon

For å tilegne seg kunnskap, skal relevant litteratur og offisiell dokumentasjon benyttes. I tilfeller der dette ikke gir tilstrekkelig informasjon, vil det i tillegg suppleres med andre kilder slik som offisielle forum og forskningsartikler som støttelitteratur.

2.1.4 Møter

Daglig Scrum møte

Hver hverdag klokken 08:15 holdes et daglig Scrum møte, hvor gruppemedlemmene oppdaterer hverandre om hva man jobber med, hva man skal gjøre videre og eventuelle utfordringer man står ovenfor.

Møte med veileder

I gjennom prosjektperioden skal ukentlige møter bli gjennomført med veileder. Før disse møtene forbereder gruppen spørsmål angående ulike problemstillinger de står ovenfor.

Møte med oppdragsgiver

Når prosjektet har beveget seg mellom ulike faser, så blir et møte med oppdragsgiver avholdt. Her gir gruppen en statusoppdatering for prosjektet så langt, og eventuelle spørsmål angånde den nye perioden oppklares. Hvis noe har vært uklart etter møte så har en Microsoft Teams kanal blitt benyttet for oppklarende spørsmål.

Oppsummeringsmøte

Gruppen har avholdt et oppsummeringsmøte periodevis, hvor den siste periodens positive og negative sider diskuteres. Dette gir gruppen mulighet til å kunne gjøre endringer og utbedringer underveis i prosjektarbeidet.

2.2 Kravspesifikasjon

I dette kapittelet vil kravene for hvordan et produksjonsklart system defineres, i tillegg til en beskrivelse av ulike use-cases systemet skal støtte.

2.2.1 Krav for å bli produksjonsklart

Når et system ansees som produksjonsklart, markeres en overgang fra en testing-fase til et system som oppfyller gitte kvalitetskrav. I denne seksjonen defineres de kvalitetskriteriene som må oppfylles før systemet regnes som produksjonsklart.

Sikkerhet

Systemet benyttes til å prosessere sensitiv informasjon. For å ivareta informasjonssikkerheten, må konfidensialiteten og integriteten av data sikres på en forsvarlig måte.

Skalerbarhet

For å få god ressursutnyttelse, så må systemet kunne skaleres opp eller ned basert på arbeidsmengden den står ovenfor.

Dokumentert

For at denne løsningen skal kunne benyttes i produksjon, kreves det dokumentasjon for hvordan systemet settes opp og brukes.

Pålitelighet

For at systemet skal kunne benyttes i operativt tjeneste, er brukerne avhengig av en løsning som fungerer som tiltenkt over lengre perioder og som er tilgjengelig ved behov. Systemet må også være lett å vedlikeholde og kunne oppdateres.

2.2.2 Use case

Use casene som presenteress er prosesser som blir ansett som essensielle funksjoner, og som ikke nødvendigvis er godt dokumentert. Dette medfører at potensiell use caser som allerede er godt beskrevet i den offisielle dokumentasjonen, som f.eks. hvordan legge til en ny bruker, ikke nevenes i denne seksjonen, men vil beskrives senere i oppgaven.

OKD use cases

Use case: 1
Navn: Sette opp en ny OKD plattform.
Aktør: System administrator.
Mål: Operasjonelt OKD kluster.

Beskrivelse: System administrator skal konfigurere og installere alle støtte tjenester som klusteret er avhengig av, og sette opp et nytt OKD kluster.

Use case: 2
Navn: Legge til ny node i OKD klusteret.
Aktør: System administrator.
Mål: Øke tilgjengelige ressurser for OKD.

Beskrivelse: Basert på monitorerings data, skaleres systemets ytelse opp ved å opprette og innrullere en ny node i klusteret.

Usecase: 3
Navn: Fjerne eksisterende node fra OKD klusteret.
Aktør: System administrator.
Mål: Redusere tilgjengelige ressurser for OKD.

Beskrivelse: Basert på monitoreringsdata, skaleres systemets ytelse ned ved å fjerne en eksisterende node fra klustret.

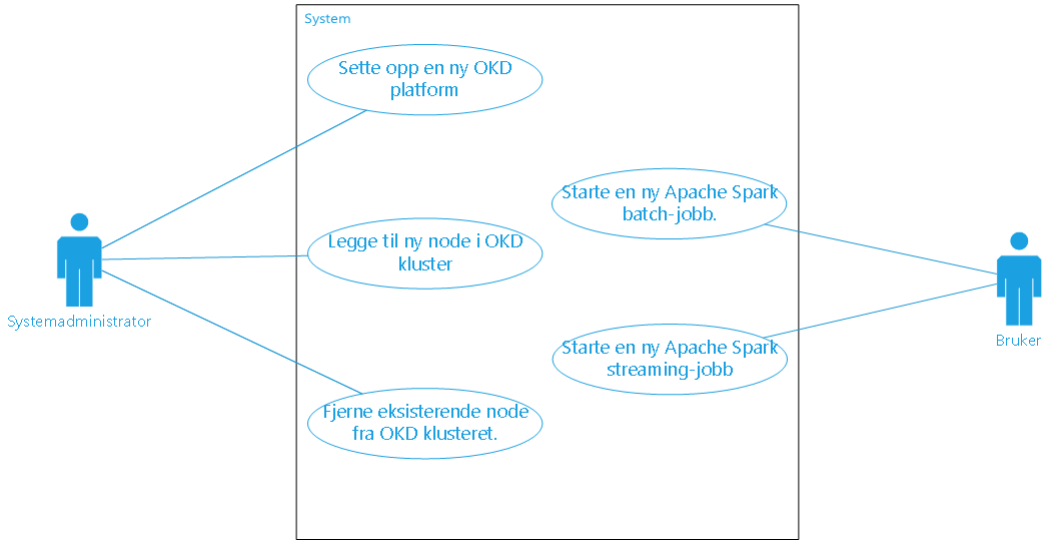
Apache Spark use cases

Use case: 4
Navn: Starte en ny Apache Spark batch-jobb.
Aktør: Bruker.
Mål: Analysere data.

Beskrivelse: Innsamlede data skal kunne sendes til Spark for analyse via en batch-jobb.

Use case: 5
Navn: Starte en ny Apache Spark streaming-jobb.
Aktør: Bruker.
Mål: Analysere data.

Beskrivelse: Innsamlede data fra ulike kilder sendes fortløpende til Apache Spark for analysering via en stream-prosess.



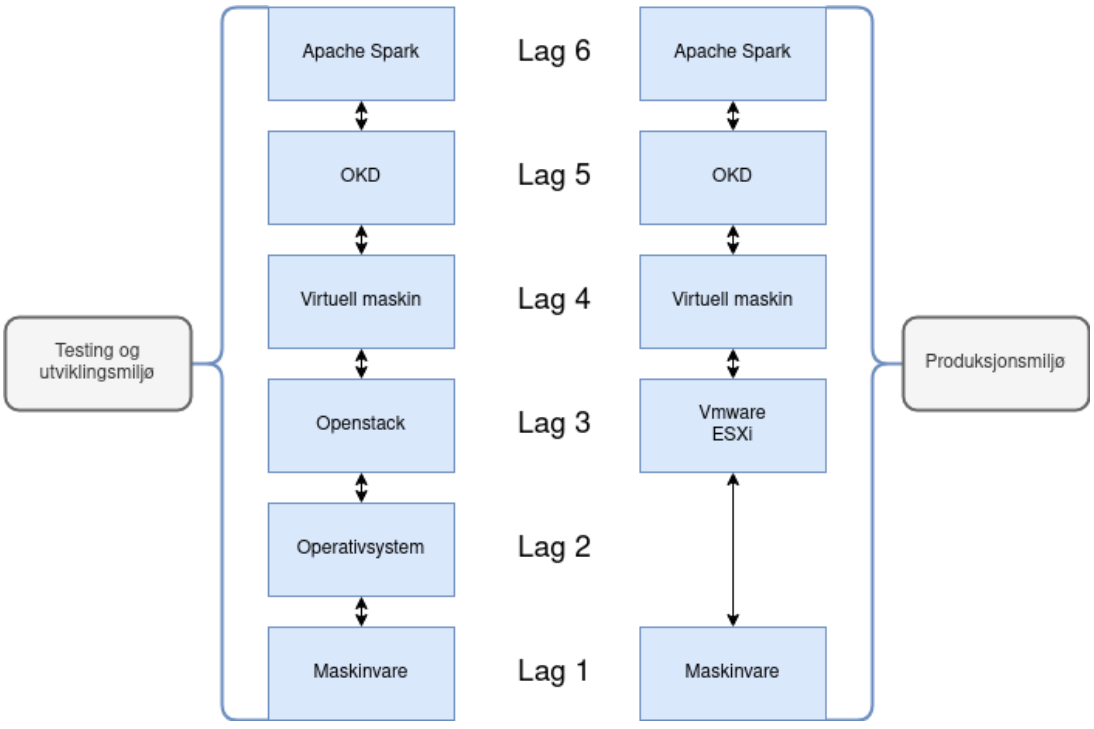
Figur 2.1: Diagram med oversikt over alle use cases og aktører i oppgaven.

Figur 2.1 er en visuell representasjon av use cases som er blitt definert i oppgaven. Det er også vist hvilke aktører som knyttet til hvilket scenario med blå piler.

Kapittel 3

Teoretisk bakgrunn

I dette kapittelet vil de mest sentrale konseptene og komponentene bli introdusert med en overordnet beskrivelse om hva det er og hvilken sammenheng de har med de andre teknologiene. Dette skaper et grunnlag for å forstå teknologiene brukt videre i rapporten.



Figur 3.1: Visualisering av de ulike komponentene i produksjonsmiljøet og testing og utviklingsmiljøet.

Kapittelet er inndelt i fire seksjoner der den første seksjonen tar for seg virtualiseringsteknologier som dekker lag tre og fire i figur 3.1. I seksjon to er lag fem beskrevet som handler om konteinere og hvordan disse blir orkestrert.

Den tredje seksjonen tar for seg stordataanalyse som dekker lag seks. Den siste seksjonen handler om distribuert lagring som blir benyttet under utviklingen og testing av løsningen.

3.1 Virtualisering

Virtualisering er en prosess som abstrakterer maskinvareressurser for å kunne dele de på flere virtuelle maskiner[3]. Det finnes to typer hypervisorer som gjør denne prosessen. Forskjellen på disse er om de er installert dirkete på hardware (Type 1) eller om det kjører som et program i et operativsystem (type 2). Hypervisoren brukt i produksjonsmiljøet er en type 1 hypervisor og heter vSphere

3.1.1 vSphere

vSphere er en virtualiseringsplattform utgitt av VMWare, som er et verdensledende firma innen virtualiseringsteknologi og har utviklet dette siden 1998[4][5]. vSphere er bygget opp av to komponenter. Den første komponenten er *VMware ESXI*, som er ansvarlig for opprettelse av virtuelle maskiner. Den andre komponenten er *vCenter server* som benyttes for å administrere maskinen hvor VMware ESXI kjører.

3.1.2 Openstack

NTNU har og drifter en privat skyløsning kalt Skyhigh. Skyløsningen er basert på Openstack som har åpen kildekode. Den tilbyr en infrastructure as a service (IaaS) tjeneste som gir brukerne tilgang til en infrastruktur[6] hvor virtuelle maskiner og nettverk kan opprettes.

3.2 Konteinerorkestrering

Før det forklares hva plattformen OKD er så, må de to kjerneteknologiene konteinere og konteinerorkestreringsverktøy introduseres.

3.2.1 Konteinere

Konteiner baserte applikasjoner har røtter helt tilbake til 1979, hvor chroot ble benyttet for å skille prosesser. Konteinere ble popularisert i 2013 da Docker kom inn på markedet. Docker, i forhold til sine konkurrenter, tilbydde et helt økosystem for administrering av konteinere som i tillegg var enkelt å bruke. Konseptet bak en konteiner er å isolere ulike prosesser så de ikke kan påvirke hverandre. Denne isolasjonen oppnås ved å benytte namespaces og cgroups for å begrense mengden resursser prosessen kan ta nytte av. For mer informasjon angående namespaces og cgroups, se avsnitt 5.2. Ved bruk av

denne teknologien kan utviklere pakke inn en applikasjon og alle avhengigheter den har i en enkelt konteiner.

Dette løser en utfordring mellom utviklere og driftspersonell, der en applikasjon fungerer fint hos utvikleren, men når applikasjonen leveres til driftspersonellet så fungerer den ikke. Grunnene til dette kan være mange, men med konteinere så er det få forhåndsregler man må ta for å distribuere dem mellom ulike miljøer.
Med denne endringen og et større fokus på smidige utviklingsmodeller begynte programutvikling å bevege seg fra å lage monolittiske systemer til å lage systemer basert på en microservice arkitektur. I en microservice arkitektur bygges systemet opp av mange små deler hvor hver enkelt del utfører en oppgave. Det oppstår alltid nye utfordringer med ny teknologi, og microservices er ikke et unntak. Ved bruk av micrservice arkitektur så vil systemet inneholde hundre eller tusenvis av konteinere som må administreres. Med tradisjonelle driftsmetoder, så blir dette en svært vanskelig oppgave. Løsningen for dette er konteinerorkestrerringsverktøy.

En av de mer kjente verktøyene på markedet i dag og det som har satt standarden for hvordan man administrerer konteinere er Kubernetes.

3.2.2 Kubernetes

Kuberentes ble utgitt av Google i 2014, og vedlikeholdes i dag av Cloud Native Computing Foundation(CNCF). Kubernetes er et rammeverk for konteinerorkestrering, og tilbyr funksjonalitet for håndtering og skalering av konteinere samt mulighet for gjenoppretting etter katastrofer. Siden Kubernetes kun er et rammeverk og ikke et ferdig produkt, så kreves det en del initiell installasjon og konfigurasjon før det kan benyttes i et produksjonsmiljø. Dette har ført til at flere bedrifter har benyttet rammeverket Kubernetes i grunn for å utvikle nye distribusjoner. En distribusjon er et ferdig produkt som kan installeres ved hjelp av et installasjonsprogram, og er da klart til bruk. Det finnes flere ulike distribusjoner av kubernetes, og en av disse er OKD.

3.2.3 Distribusjonen OKD

OKD/Openshift er Red Hat sin distribusjon av Kubernetes. Openshift er en betalt versjon som kommer med brukerstøtte, mens OKD er et gratis alternativ uten brukerstøtte.
I motsetning til kubernetes, som kun gir tilgang til konterinerokistreringsverktøy, så er OKD et ferdig oppsatt produkt som er klar til bruk. Den kommer med et oppsatt grafisk grensesnitt og har et større fokus på sikkerhet med strengere definerte roller, og benytter *non-root* konteinere som standard. For mer informasjon angående non-root konteinere, se 5.3.1.

3.3 Stordataanalyse

Stordata beskrives ved de tre store V'ene, Volume, Variety og Velocity. Dette kjennetegnes ved at det kommer i stort volum, som oftest større en en terrabyte [7]. Stordata kommer i en bred variasjon av formater fra ulike kilder og kan være både strukturert og ustrukturert data. Det kjennetegnes også ved at det både produseres og er tilgjengelig mye raskere enn før[8].
Stordataanalyse er relativt likt vanlig dataanalyse, men når stordata skal analyseres, kreves andre løsninger som håndterer de store datamengdene som behandles i systemet[9]. Et slikt verktøy er Apache Spark.

3.3.1 Apache Spark

Hadoop Mapreduce er forgjengeren til Apache Spark. Dette er et rammevek for å skrive applikasjoner som kan håndterer store mengder data parallelt. Det kan brukes på en enkelt node men også skaleres opp til å kjøres distribuert. En utfordring med Mapreduce, er at den må hente og skrive data til disk for hver enkelt operasjon. På grunn av blant annet denne begrensningen, begynte Berkeley's AMPLab[10] utviklingen av et nytt og bedre analyseverktøy. Analyseverktøyet, kjent som Spark, ble lansert i 2014 og deretter donert til *non-profit* orginisasjonen *Apache Software Foundation*[10] for administrering og videreutvikling. Apache Spark løser problemet vedrørende ytelse MapReduce har ved bruk av Resillient Distributed Datasets (RDD) som lagres i minnet.

Spark er en generell analysemotor som kan benyttes på enkeltmaskiner og i større kluster hvor analysen kjøres distribuert. Det er bygget for å være enkelt å bruke, effektivt og fleksibelt[11], med støtte for flere anerkjente programmeringsspråk som Scala, Python, R og Java. I tillegg til dette, så har spark støtte for å hente data fra ulike plattformer[1] som Apache Cassandra, Apache HBase, Alluxio og Hadoop File System (HDFS).

Spark kan utføre jobber på to ulike metoder, ved batch eller stream prosessering. I en batch jobb så analyseres allerede innsamlet data i bulk. I noen tilfeller så kan det være nyttig å analysere data fortløpende ut fra innkommende informasjon. For denne type analyse benyttes stream prosessering.

3.4 Distribuert lagring

For å oppbevare data som innsamles er det behov for store mengder lagringsplass. En måte dette kan gjøres på er å benytte seg av distribuert lagring. I distribuert lagring kobles flere noder sammen til et kluster for å fordele data ut over de ulike lagringsenhetene. Fordelen med å lagre data spredt er at det gir muligheten til å replikere data, sørge for bedre feilsikring

[1] https://spark.apache.org/

og redundans. Om problemer skulle oppstå med en av nodene i klusteret, er det fremdeles mulig å hente ut fullstendig data ettersom det er replikert til flere enheter. Det er også skalerbart ved at man kan koble til flere noder for å øke lagringskapasiteten. Et eksempel på et distribuert filsystem er Hadoop File System (HDFS).

3.4.1 HDFS

HDFS er en type distribuert lagring som er underlagt Apache Hadoop økosystemet. Filsystemet er designet for å operere på maskinvare tilgjengelig for forbrukere[12], og er bygget for å være feiltolerant samt levere høy ytelse. Data blir lagret i store blokkstørrelser, noe som gjør systemet godt egnet for å lagre store filer. Dette er attributter som gjør HDFS til en attraktiv lagringsløsning å kombinere med blant annet verktøy for stordataanalyse.

Kapittel 4

Designvalg og konfigurasjon av Apache Spark på OKD

4.1 Apache Spark

Apache Spark er i mange selskaper et viktig verktøy brukt av blant annet av Netflix og Facebook.[11]. Denne seksjonen går i gjennom momenter ved en konteinerbasert utgave av Apache Spark, og hvordan en bruker interakterer med Apache Spark satt opp i kluster.

4.1.1 Apache Spark i konteiner

Å få pakket Apache Spark i en konteiner er et krav for å få programmet til å kjøre i Kubernetes. Dette medbringer fordeler, ulemper og noen prosedyrer som må gjøres annerledes enn om den hadde kjørt i et standard miljø uten bruk av konteinere.

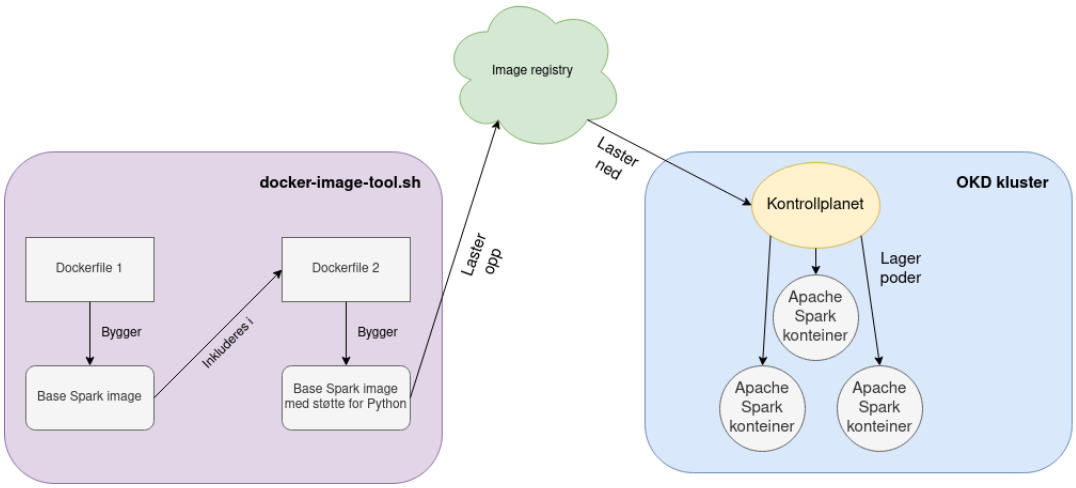
Bygge image av Apache Spark

Selskaper som Google, Bitnami med fler, gir ut egne versjoner av Apache Spark i konteiner. Bitnami er en annerkjent aktør eid av VMware og har over en million nedlastninger på sitt Apache Spark Docker image fra Dockerhub[1]. Fordelen ved bruk av disse imagene er at de blir hyppig oppdatert og har en stor brukerbase.

Ved å bruke et image laget av en annen aktør er det ikke sikkert man får den funksjonaliteten eller at det dekker de kravene satt for applikasjonen. Apache Spark tilbyr to forskjellige metoder for å lage et eget image. Det første alternativet er ved bruk av Apache Maven som ikke vil bli dekt i denne rapporten. Det andre alternativet er ved bruk av et innebygd verktøy kalt

[1] https://hub.docker.com/r/bitnami/spark

docker-image-tool.sh. Dette er et shellscript som bygger et image ved hjelp av Docker sin daemon.



Figur 4.1: Visuell representasjon av hvordan image bygd av docker-image-tool.sh blir brukt i OKD.

Docker-image-tool.sh benytter to Dockerfiler for å bygge et image som har støtte for Python. Den første er ansvarlig for å opprette et base image. Base imaget installerer nødvending programvare og Apache Spark, kopierer over filer og til slutt endrer bruker id i konteineren for å være kompatibel med OKD sine retningslinjer om å ikke kjøre konteiner som root. Se mer om dette i kapittel 5.2. Den andre Dockerfilen bygge som et lag på toppen av base image og legger til støtte for Python før den til slutt endrer brukerid som i base image. Etter at imaget er opprettet, kan docker-image-tool.sh også benyttes for å pushe det ferdige imaget til et image registry hvor Kubernetes operatoren eller spark submit kan bruke det for å utrullere et Apache Spark kluster i OKD.

Håndtering av avhengigheter i konteinere

Det er flere løsninger på hvordan man håndterer avhengigheter generelt for et program i en konteiner. En mulighet er å lage et image som inneholder alle avhengigheter til alle typer applikasjoner man skal bruke. Fordelen med dette er at da kan man bruke samme image for alle applikasjoner. En konsekvens av dette er at imaget blir stort og over lengre tid vil dette belaste nettverksinfrastrukturen ved hyppig nedlasting av image. En annen ulempe er at om en applikasjon krever en bestemt versjon av en avhengighet, mens en annen applikasjon krever en annen, kan ikke samme image brukes for begge. Dette er en stor ulempe som ikke går overens med tankegangen om å bruke samme image for alle applikasjoner.

En annen løsning er å lage et *base image* der avhengigheter blir lagt til for

hvert enkelt scenario som inneholder kun de avhengighetene applikasjonen trenger. Dette er funksjonalitet Docker kan bidra med ved at Docker bygger nye lag oppe på allerede eksisterende konteinere slik at hele prosessen for å bygge et image ikke trengs å gjøres for hver gang et nytt lag legges til.

4.1.2 Interaksjon mellom bruker og Apache Spark kluster i OKD

Spark submit er et av verktøyene som følger med når Apache Spark lastes ned. Dette verktøyet brukes for å interaktere med Apache Spark klusteret, og vil være oppdatert og støtte samme versjon, som på nåværende tidspunkt er versjon 3.1.1. Dette er den offisielle måten å utrullere Spark applikasjoner på.

En alternativ måte å interaktere med klusteret på, er ved bruk av en Kubernetes operator. En Kubernetes operator er en applikasjonsspesifikk kontroller som automatiserer og setter opp en applikasjon I OKD. OKD er koblet opp mot Red Hat Marketplace, som fungerer som et applikasjonsbibiliotek der brukere kan installere programvare administrert av Red Hat[2]. Det finnes to Apache Spark operators på Marketplace i OKD. En operator laget av Radanalytics[3], og en fra Google Cloud Computing (GCP)[4]. Radanalytics er en gruppe der flere av medlemmene jobber for Red Hat som har utviklet en egen Apache Spark Kubernetes operator. Den andre operatoren er laget av GCP som er Google sin egen avdeling for utvikling av skyløsninger, og denne går under navnet spark-on-k8s-operator. GCP sin spark operator er i beta, men blir likevel brukt av selskap som CERN, UBER og MongoDB. MongoDB bruker den i sitt produksjonsmiljø[5].

Bruk av ulike Apache Spark versjoner

Begge Kubernetes operatorene tilgjengelig på Red Hat Marketplace støtter versjon 2.4 av Apache Spark, men GCP støtter i tillegg versjon 3.0 om det blir installert via Helm sine Helm charts tilgjenglig på GCP sin Github[6].
Apache Spark fikk native støtte for kubernetes i versjon 2.3 og har siden da lagt til mer funksjonalitet. Versjon 3.1.1 er den første versjonen som er markert som *generally available* og som dermed er produksjonsklar[7]. I versjon 3.0 ble dynamisk tildeling av ressurser mulig uten bruk av en ekstern tjeneste. Dette er et steg mot komplett dynamisk skalering av Apache Spark der dynamisk allokering vil bli testet senere i kapittel 6.

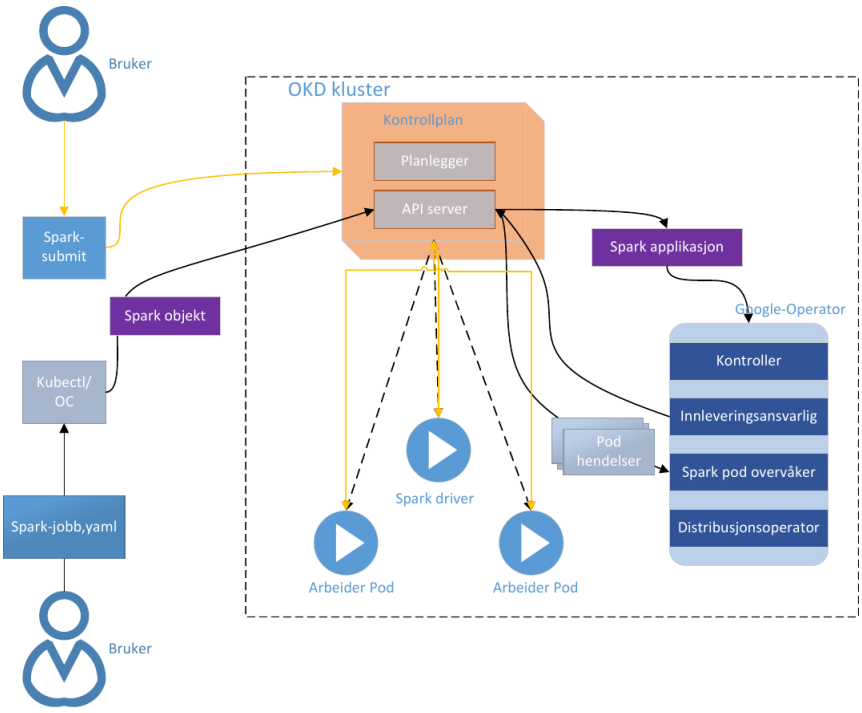
[2]https://docs.okd.io/latest/applications/red-hat-marketplace.html
[3]https://github.com/radanalyticsio/spark-operator
[4]https://github.com/GoogleCloudPlatform/spark-on-k8s-operator
[5]https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/blob/master/docs/who-is-using.md
[6]https://github.com/GoogleCloudPlatform/spark-on-k8s-operator
[7]https://spark.apache.org/releases/spark-release-3-0-0.html

Valg av spark operator

Gruppen kommer til å bruke GCP sin spark operator videre i oppgaven og anbefaler denne fremfor Radanalytics sin spark operator, da den er mest oppdatert og har mest dokumentasjon for å sende Apache Spark jobber til klusteret. Alle eksempler og konfigurasjon videre i rapporten som har med bruk av spark operator vil være basert på GCP sin spark operator.

Arkitektur

Både Apache Spark operator og spark submit benytter de samme funksjonene ved å kontakte kubernetes sin API server som scheduler en driverpod. Driverpoden snakker med scheduleren og avtaler hvor mange arbeidspoder den trenger. Dette kan skje statisk eller dynamisk ved å benytte dynamisk allokering, som demonstreres i kapittel 6. Hovedforskjellene mellom disse metodene er at Apache Spark operatoren lager applikasjonen som et Kubernetes objekt som kan konfigureres via YAML fil, og har innebygd støtte for monitorering.

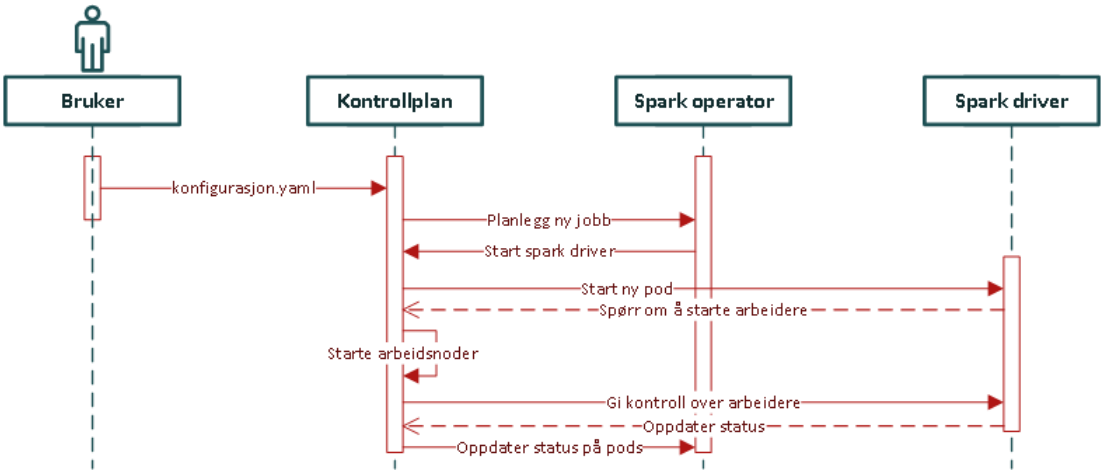


Figur 4.2: Arkitektur for Apache Spark på OKD, basert på Google-cloud operatoren [8]. Gule linjer viser spark-submit metoden for å stare jobber. Svarte linjer viser sammenhengen mellom komponenter ved bruk av GCP operator.

[8] https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/blob/master/docs/design.md

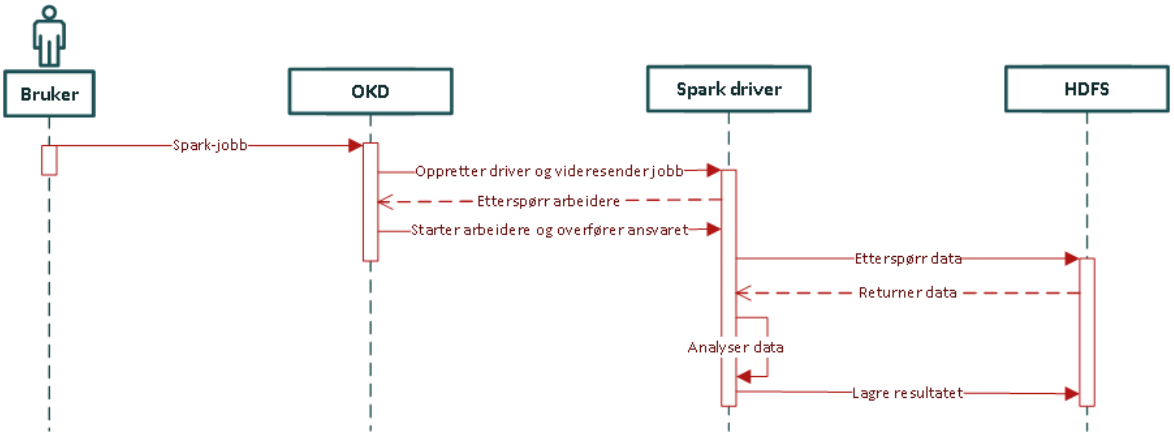
Figur 4.2 beskriver arkitekturen og viser hvilke komponenter som interakterer med hverandre ved start av Apache Spark jobber på OKD. Illustrasjonen viser to metoder for å sende jobber til et OKD kluster. Via *spark-submit* kommandoen blir jobben sendt til OKD klusteret hvor en kontrollplan node starter spark pod for å gjennomføre den innsendte jobben. Google-cloud operatoren blir initialisert ved å sende en spark-konfigurasjonsfil til OKD klusterert med terminal-klienten *Kubectl/oc*. Kontrollplanet videresender spark objektet som ble laget av *kubectl* til Google-operatoren som svarer med antall arbeidspoder som skal lages og hvilke innstillinger som skal benyttes.

Sekvensdiagram



Figur 4.3: Sekvensdiagram for å beskrive hvordan Google-cloud spark operatoren interakterer med klusteret. Heltrukne linjer symboliserer interaksjon, striplete linjer er svar tilbake.

Dette sekvensdiagrammet er en visuell fremstilling som viser hvilke interaksjoner det er mellom brukeren, OKD og spark operatoren. Den sekvensen som vises er ved bruk av en Google cloud-operatoren som er beskrevet i forrige figur. Det hele initialiseres med at brukeren sender en konfigurasjonsfil til OKD via programmet *kubectl/oc*. Kontrollplanet videresender jobben til Spark operatoren som formaterer konfigurasjonsfilen og sender den til kontrollplanet i OKD med beskjed om å starte en spark driver. Spark driveren sier så til kontrollplanet hvor mange arbeidere som skal startes basert på konfigurasjonen. Jobben blir da gjennomført på arbeiderne med driveren som organiserer arbeidet. Spark-operatoren blir også oppdatert på statusen til podene fra kontrollplanet.



Figur 4.4: Sekvensdiagram for å vise hvordan en spark-jobb kjører. Heltrukne linjer symboliserer interaksjon, striplete linjer er svar tilbake.

Dette sekvensdiagrammet er et forstørret utdrag fra figur 4.3. Denne figuren er en grafisk presentasjon av hvordan selve spark-jobben blir gjennomført og inkluderer ikke like mye overordnet hvordan systemet fungerer. Her er også spark operatoren utelatt fra diagrammet da det ikke er like essensielt for hvordan selve analyse-jobben blir gjennomført og gjør at diagrammet kan representere begge metodene å kjøre spark-jobber på. Som forklart tidligere blir prosessen initialisert av bruker som sender en spark-jobb til OKD. Et kontrollplan i OKD vil så opprette en spark-driverpod og sende med jobben den fikk levert. Spark-driveren vil tolke jobben den fikk og be kontrollplanet starte det antallet arbeidere som er spesifisert. Når spark-driveren får arbeiderpod ene blir de satt i gang med å hente og analysere data fra HDFS som spesifisert i jobben. Arbeidet utføres av arbeidspodene med spark-driveren som administrerende. Da analysejobben er fullført blir resultatet lagret tilbake i HDFS.

4.2 Hadoop Distributed File System

Etter samtaler med oppdragsgiver ble HDFS diskutert som en mulig lagringsløsning da de allerede er kjent med teknologien og benytter det i dag. HDFS er tett integrert med Apache Spark, og begge teknologiene støttes av hverandre uten behov for videre konfigurasjon. Siden oppdragsgiver er godt kjent med HDFS er det bare behov for å ha et enkelt oppsett i utvikling og testingsinfrastrukturen for å teste kommunikasjon med Apache Spark, og lagring av data for bruk med Spark applikasjoner. Det tas utgangspunkt i at oppdragsgiver konfigurerer HDFS for sitt bruksområde og det vil derfor ikke bli gjennomgått i dypden i rapporten.

HDFS er et filsystem som normalt kjøres distribuert over flere noder og består at en namenode og opptil flere datanodes. Namenode har ansvar for å regulere filaksess og lagrer metadata om filene distribuert rundt på de ulike namenodene.

Det kan settes opp på en enkelt node i enten *standalone operation*, der HDFS kjørere i en enkel java prosess, eller i *pseudo-distributed operation*, der HDFS simulerer at den kjøres over flere noder ved å benytte seg av flere java prosesser [9].

I oppgaven blir *pseudo-distributed operation* benyttet da dette er mest likt hvordan HDFS oppfører seg distribuert uten å måtte sette av ressurser til flere noder. Dette er ikke et oppsett som anbefales å benytte i produksjoninfrastrukturen, men oppfyller sin hensikt for test av systemets integrasjon med Apache Spark. For å sette opp en pseudo-distribuert HDFS instans brukes den offisielle dokumentasjonen fra Apache[10].

Installasjon

HDFS er et Java program og er avhengig av en støttet versjon på noden det skal installeres på for å fungere. HDFS ble testet med versjon 8 og 11 av Java, hentet med Fedora sin packet manager dnf. Det oppsto med begge disse versjonene problemer med HDFS noe som ble løst ved å fjerne eksisterende versjoner for å så installere *open java development kit (openjdk)* versjon 8[11].

Videre i installasjonsprosessen kreves det å sette opp passordløs ssh for å kunne kommunisere med de ulike HDFS komponentene. Etter at passordløs ssh er konfigurert, kan man teste om det er satt opp riktig med kommandoen *ssh localhost*. Hvis man får feilmeldingen *"Connection refused"*, er det to mulige løsninger. Første løsningen er å bytte ut påfølgende kodelinje i filen "$HADOOP_HOME/libexec/hadoop-functions.sh".

```
if [[ -e '/usr/bin/pdsh' ]]; then
```

Kodeliste 4.1: Gammel kode i hurtigløsning i HDFS installasjon.

Med:

```
if [[ ! -e '/usr/bin/pdsh' ]]; then
```

Kodeliste 4.2: Ny kode for hurtigløsning i HDFS installasjon.

Første midlertidige løsning vist i kodeliste 4.2 gjør at HDFS fortsetter uten å benytte pdsh.

[9] https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html#Standalone_Operation

[10] http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

[11] https://www.openlogic.com/openjdk-downloads

Den andre løsningen er å bytte ut følgende kodelinje i filen $HADOOP_HOME/libexec/hadoop-functions.sh.

```
PDSH_SSH_ARGS_APPEND="${HADOOP_SSH_OPTS}" pdsh \
```

Kodeliste 4.3: Gammel kode for å fikse installasjon i HDFS.

Med:

```
PDSH_RCMD_TYPE=ssh PDSH_SSH_ARGS_APPEND="${HADOOP_SSH_OPTS}" pdsh
    \
```

Kodeliste 4.4: Ny kode for å fikse installasjon i HDFS.

Den andre løsningen vist i kodeliste 4.4, er litt mer elegant og anbefalt, siden den sørger for at videre kode benytter pdsh men med ssh i stede for rsh.

Dette er et problem som oppstår ved at scriptet benytter kommandoen pdsh for å kjøre kommandoer på flere andre maskiner samtidig. Pdsh forsøker å bruke et eldre program rsh i stede for ssh til å gjennomføre kommandoer som igjen sørger for problemer med å koble til localhost[12]. Midlertidige løsninger er funnet og rapportert til utviklerne sammen med en Jirasak med saksnummer *HADOOP-15219*[13].

Konfigurasjon

Det kreves konfigurasjon før HDFS kan benyttes. I denne sammenheng oppstår en utfordring med lagringsplass HDFS har tilgang til. For å løse dette problemet, må konfigurasjonen oppdateres for at HDFS skal benytte en annen mappe enn /tmp. Denne mappen er satt som standardlokasjon for lagring, og har en begrenset mengde med tilgjengelig lagringsplass. I filen *core-site.xml* oppdateres *value* med absolutt stien til mappen HDFS skal benytte[14].

```
<property>
    <name>hadoop.tmp.dir</name>
    <value>/home/myUserID/hdfs</value>
    <description>base location for other hdfs directories.</
        description>
</property>
```

Kodeliste 4.5: Konfigurasjon for å øke lagringsplass i HDFS

4.2.1 Egnethet for batch og stream prosessering

HDFS er et filsystem som er designet for å lagre store filer. Filsystemet bruker store blokkstørrelser der 128MB er satt som standard. Formålet ved å benytte

[12] https://stackoverflow.com/questions/48189954/hadoop-start-dfs-sh-connection-refused
[13] https://issues.apache.org/jira/browse/HADOOP-15219
[14] https://stackoverflow.com/questions/19542185/how-can-i-increase-hdfs-capacity

en stor blokkstørrelse er at mer av diskplassen kan benyttes til lagring av data, og ikke header informasjon om dataen. HDFS er laget for å prioritere høy datagjennomstrømning fremfor lav responstid ved aksesserering av filer [15].

HDFS er ikke like egnet for å håndtere streaming, da denne metoden produserer i større grad flere små filer enn batch jobber. Namenoden i HDFS holder oversikt over blokker og filer i ram. Ved å ha et høyt antall filer krever dette mye ram for å ha oversikt over alle filene på engang [13].

4.3 Ulike installasjonsmetoder for OKD

Etter planlegging og møter med oppdragsgiver ble det fastsatt at utvikling og testing av løsningen skal foregå i skytjenesten til NTNU, som benytter Openstack, mens det ferdige produktet skal operere i et vSphere miljø. OKD har støtte for flere ulike installasjonsmetoder for begge disse plattformene, som kan deles opp i to hovedkategorier kalt *Installer provisioned infrastructure (IPI)*, og *user provisioned infrastructure (UPI)*.

Forskjell mellom UPI og IPI

Hovedforskjellen mellom en UPI og en IPI installasjon er hvordan de ulike komponentene i OKD klusteret opprettes før, under og etter installasjonsprosessen. Når en UPI installsjon benyttes, så er man avhengig av å manuelt sette opp alle komponentene OKD trenger. Dette innebærer maskinene som klusteret skal bestå av, nettverksinfrastruktur, operativsystem og støttetjenester. En UPI installasjon kan enkelt integreres i en allerede eksisterende infrastruktur, og benytte tjenester som allerede eksisterer.
En IPI installasjon automatiserer flere av stegene som må gjøres manuelt i en UPI installasjon. For å gjøre dette, så behøver installasjonsprogrammet tilgang til nødvendige rettigheter for å kunne opprette de ulike komponentene. Dette gjør installasjonsprosessen enklere, og mindre konfigurasjon er påkrevet før installasjonsprosessen kan starte.

For å kunne starte med testing og utforsking av OKD, så benyttes en IPI installasjon i NTNU sin skytjeneste Skyhigh. Ved å benytte denne metoden, så tilrettelegges det for automatisk skalering av virtuelle maskiner.

Sluttproduktetet skal brukes i vSphere, som er en av de offisielt støttede plattformene for OKD. I følge installasjons dokumentasjonen for OKD i vSphere krever dette tilgang til *VMware NSX-T*. Dette er lisensbelagt funksjonalitet som ikke er et alternativ å benytte i dette prosjektet. For å kommer rundt denne avhengigheten må en installasjonsmetode som ikke krever tilgang til lisensbelagt funksjonalitet benyttes. Dette kan utføres ved å

[15] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

simulere en *bare metal* installasjon med bruk av virtuelle maskiner i vSphere. Ved å benytte virtuelle maskiner, så vil kluster kunne operere i flere ulike virtualiseringsmiljøer. Dette er mulig siden alle støttetjenester installeres og konfigureres manuelt, og OKD sitt installasjonsprogrammet kun benyttes for å generere nødvendige konfigurasjonsfiler.

Baremetal har enda en offisiell installasjonsmetode som er beregnet for fysiske servere, hvor man benytter seg av en *Baseboard Management Controller (BMC)* for å konfigurere serveren. Siden sluttproduktet skal operere i vSphere på virtuelle maskiner, regnes denne metoden som uegnet for dette prosjektet. Ut i fra disse ulike aspektene som er nevnt, vil en versjon av installasjonsmetoden *bare metal UPI* benyttes.

4.4 Installasjon av OKD

I denne seksjonen vil en utvidet beskrivelse av installasjonsmetoden, bare metal UPI, presenteres. Det vil bli gitt en oversikt over påkrevde støttetjenester, hvorfor disse er i bruk, og hvordan de er konfigurert. For oversikt over spesifikke kommandoer som viser steg for steg installasjon se vedlegg E.1.
Installasjonen er basert på den offisielle dokumentasjonen til OKD utgitt av RedHat, og arbeid utført av *Craig Robinson*[16] [17].

4.4.1 Beskrivelse av miljø

Installasjon, oppsett og konfigurasjon har blitt utført på en stasjonær pc med en Ryzen 7 2700x CPU, 64GB DDR4 og en 512GB M.2 SSD.
Bruk av en 7200 RMP HDD på 2TB ble først brukt for installering, men initialisering av klusteret ble aldri fulført da bootstrapnoden og kontrollplannodene mislykkes i å opprette et ETCD kluster. ETCD er sensitiv på treg I/O derfor antas det at hastigheten på HDD benyttet var årsaken til at det ikke fungerte[18].

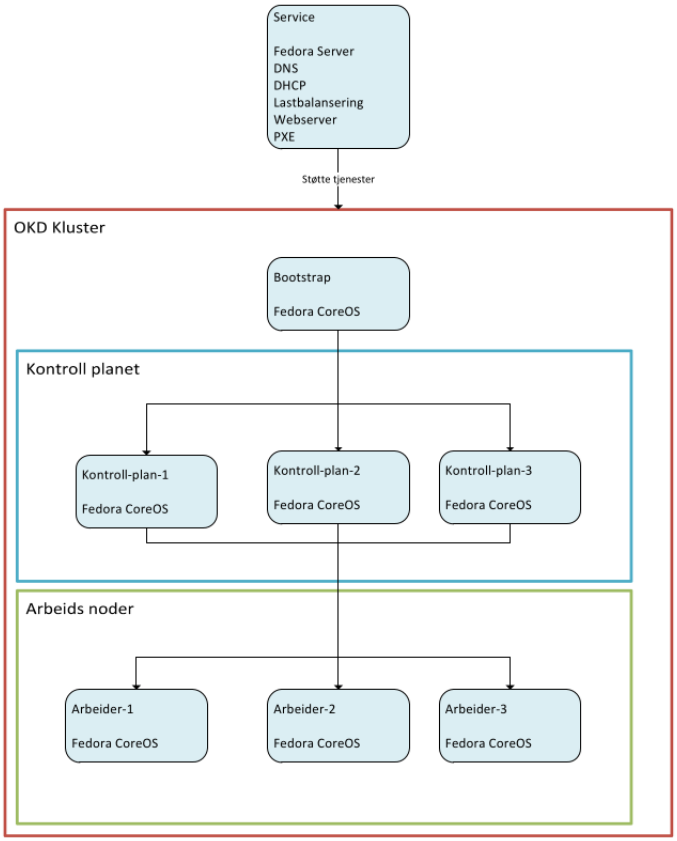
Det kan ikke utelukkes at problemer og feil som er beskrevet i rapporten ikke skyldes maskinvare, da dette nødvendigvis ikke er representerbart for en infrastruktur et OKD kluster er laget for å operere i.

4.4.2 Overordnet installasjonsbilde

For å opprette et OKD kluster, så er nodene avhengig av informasjon om klusteret. Dette gjøres ved å benytte den midlertidige noden, *bootstrap*, under initiell konfigurasjons av kontrollplanet. Den initielle konfigurasjonen er også

[16] https://github.com/cragr
[17] https://itnext.io/okd-4-5-single-node-cluster-on-windows-10-using-hyper-v-3ffb7b369245
[18] https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#prerequisites

kjent som bootstraping. I denne prosessen så spesifiseres det hvilken noder som tilhørere kontrollplanet og hvilken noder som er arbeidsnoder. Informasjon om hvordan klusteret skal konfigureres beskrives i ignition filer. Disse ignition filene genereres av OKD sitt installasjonsprogram, og er spesifikk for node typen. Dette vil si en igniton file for bootstrap, kontrollplan og arbeidsnodene. Når kontrollplanet er satt opp kan bootstrapnoden fjernes, og kontrollplanet konfigurerer deretter arbeidsnodene.

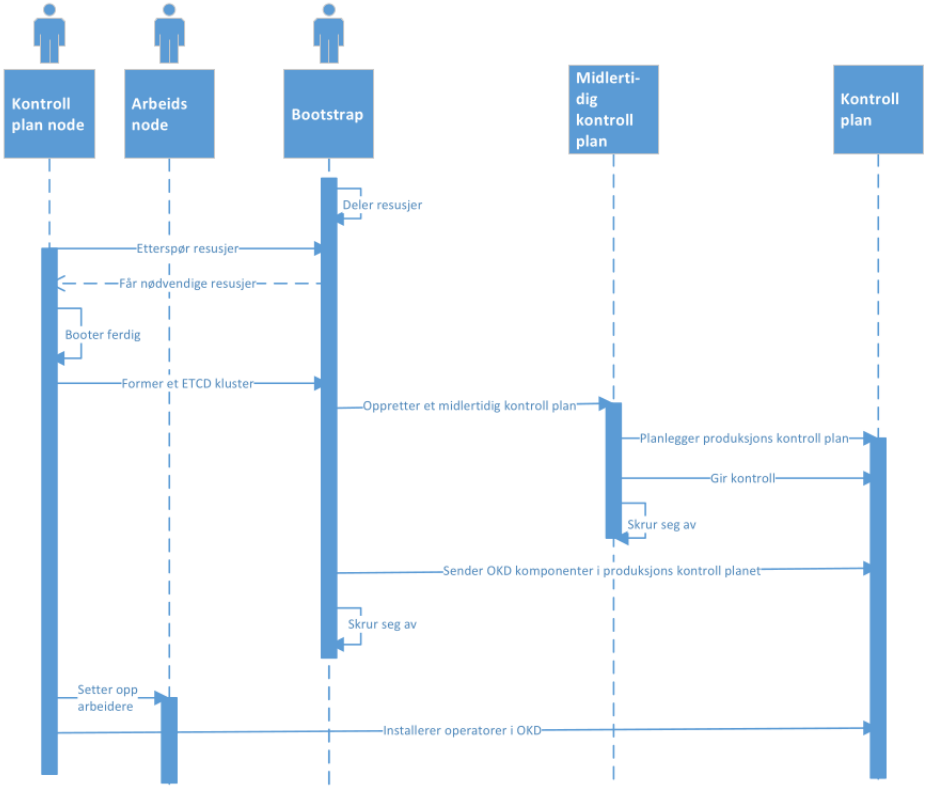


Figur 4.5: Oversikt over hvordan de ulike komponentene henger sammen.

Bootstrap prosessen

Bootstapping av et kluster starter med at bootstrapnoden henter ignition filen og deler resursene som kontrollplanet trenger. Kontrollplanet tar så å henter de nødvendige ressurser og starter opp.
Kontrollplannodene benytter så bootstrap noden til å opprette et etcd kluster. Etcd er hovedlagringsmekanismen for Kubernetes, der den replikerer status for

alle komponentene i klusteret. Bootstrapnoden starter så et midlerditig Kubernetes kontrollplan for kontrollplannodene. Det midlertidige kontroll planet planlegger det som skal bli det permanente kontrollplanet som skal benyttes av klusteret. Når dette er gjennomført skrur det midlertidige kontrollplanet seg av og gir kontrollen over til det permanente kontrollplanet. Bootstrapnoden tar så å injiserer OKD komponenter inn i det nye kontrollplanet, og skrur seg av når dette er gjennomført. Kontrollplanet setter så opp arbeidsnodene, og installerer tilleggstjenester som klusteret er avhengig av.
Etter disse stegene er gjennomført, så har man et fungerende OKD kluster.



Figur 4.6: Sekvens diagram som viser stegene noden tar for å initialisere det permanente klusteret.

4.4.3 Konfigurasjon av noder

Denne seksjonen beskriver designvalg av hvilket operativsystem som skal benyttes, størrelse på nodene og hvor mange noder som skal benyttes beskrives. Kravene som er satt kommer fra den offisiell dokumentasjon eller fra oppdragsgiver.

Operativsystem

Hvilken operativsystem som skal benyttes, er bestemt ut i fra to kriterier. Den første er et krav fra oppdragsgiver at det skal ha støtte for Security Enhanced Linux (SELinux). Dette er en sikkerhetsmodul i Linux kernelen som gir bedre kontroll over hva som kan få tilgang til systemet. Mer informasjon angående SELinux er beskrevet i kapittel 5.2.1. Det andre er anbefalinger gitt fra den offisielle dokumentasjonen til OKD.
Dokumentasjonen anbefaler *Redhat Enterprise Linux*(RHEL) eller *Fedora* som mulige kandidater. RHEL er et lisensbelagt produkt som distribueres av RedHat, og Fedora er *upstream* versjonen av RHEL. Begge har SELinux innebygd, og har versjoner som er spesialisert for å kjøre konteinerbaserte oppgaver. Siden lisens kreves for å bruke RHEL, så vil to ulike versjoner av Fedora brukes. *Fedora CoreOS*(FCOS) brukes på klusternodene, og er et minimalistisk konteinerfokusert operativsystem optimalisert for bruk sammen med Kubernetes[14]. For noden der støttetjenestene kjøres, brukes Fedora Server.

Versjoner av operativsystem

For produksjonsmiljø er det ønskelig med bruk av stabile versjoner av programvare for å sørge for høy oppetid og at komponenter fungerer som de skal. OKD har ingen *long term support* versjon, da produktet fungerer som en upstream versjon av Red hat Openshift. Det har blitt testet ulike versjoner og kompatibilitet mellom disse for å kunne anbefale hvilke versjoner som egner seg for installering.

FCOS utgis omtrent en gang hver sjette måned[19], og har alltid to versjoner ute som blir vedlikeholdt. På nåværende tidspunkt er FCOS 32 og FCOS 33 de to versjonene som er støttet, og de to potensiale kanditatene for operativsystem brukt i konteinerne. Under utrulleringen av klusteret, oppstod noen kompatibilitetutfordringer mellom installasjonsprogrammet til OKD og ulike versjonen av FCOS.

[19]https://fedoraproject.org/wiki/Fedora_Release_Life_Cycle#Release_Dates

Test av OKD 4.7 og FCOS 33

Installasjonsprogrammet med versjonen 4.7 har støtte for FCOS 33, men for å kunne starte installasjonsprosessen så mangler programmet *OC*. OC er Openshift sitt verktøy for a interaktere med OKD, og er basert på Kubernetes sitt verktøy kalt *kubectl*. Man skal egentlig ikke trenge OC kommandoen for å starte installasjonsprosessen, siden den skal falle tilbake til å benytte podman, som den gjør, men hvis OC ikke er tilgjengelig så er det en kommando som feiler litt senere.

Test av OKD 4.6 og FCOS 33

For installasjonsprogram versjon 4.6, så er ikke FCOS 33 støttet, og ved å benytte FCOS 32 oppstår problematikken som i versjon 4.7. En mulig løsning på dette, er å kopiere OC kommandoen til */usr/local/bin/* før installasjonsprosessen starter. Dette kan løses ved å benytte *SCP* for å kopiere filen, og ssh for å koble til node og flytte filen til riktig plass. Dette må skje med en gang ssh er tilgjengelig. Dette gjør det upraktisk å utrullere, og selv om man er rask nok. Etter testing av denne løsningen på problemet så har antall suksessfulle utrulleringer vært få i forhold til antall mislykkede og kan ikke anses som en pålitelig måte å gjøre dette på.

Test av OKD 4.5 og FCOS 32

Med installasjonsprogram versjon 4.5 og FCOS 32 har vi hatt bedre hell med. Med denne kombinasjonen har vi, etter at støttetjenester og konfigurasjons filer er oppretter, automatisert utrulleringen og man trenger ikke å gripe inn under prosessen. Dette betyr ikke at utrulleringen alltid fullføres, men det skjer oftere enn at den misslykkes.

OKD er under kontinuerlig utvikling, og det antas at bedre kompatibilitet mellom ulike FCOS versjoner og installasjonsprogrammet vil bli utgitt i fremtiden.

Bruk av golden image

Når man jobber med utrullering av infrastrukturer, er forhåndskonfigurerte operativsystemer utbredt. Disse inneholder ofte grunnleggende konfigurasjon og programmer bedriften bruker. Dette er også kjent som et *Golden image*. I dette prosjektet fungerer dette fint for noden som er ansvarlig for støttetjenester, men det egner seg ikke for klusternodene. Under installasjon av FCOS benyttes en ignition file. Denne filen er generert av installsjonsprogrammet til OKD, og inneholder klusterspesifikk informasjon. En av disse er et sertifikat som

kun er gyldig i 24 timer, som gjør det upraktisk å benytte seg av en Golden image for FCOS da et golden image ikke vil være brukbart etter 24 timer har passert. For å komme rundt denne utfordingen, benyttes et Preboot Execution Environment (PXE) i steden for en golden image. PXE tilrettelegger for å kunne starte en installasjonsprosess over nettverket, og er basert på MAC-addressen til en node, som bestemmes hvilken ignition filer den skal hente og hvor operativsystemet den skal benytte ligger. Denne metoden er beskrevet mer i dybden i avsnitt 4.4.4.

Krav for maskinvare

OKD krever et visst antall ressuser for å operere stabilt. Maskinvareressursene som tildeles de forskjellige nodene er basert på anbefalinger gitt av Red Hat. Ut i fra dette er følgende minimumskrav satt[15].

Node	Operativsystem	vCPU	Ram	Lagring
Bootstrap	FCOS	4	16 GB	120 GB
Kontrollplannode	FCOS	4	16 GB	120 GB
Arbeidsnode	FCOS	2	8 GB	120 GB
Servicenode	Fedora server	4	4 GB	100 GB

Tabell 4.1: Maskinvarekrav for klusternodene.

vCPU er virtuelle CPU kjerner. Dette vil si om man utnytter hyper-threading (Intel) eller Simultaneous multithreading (AMD) så gir 1 CPU kjerne 2 vCPU kjerner, om dette ikke blir benyttet så telles 1 CPU kjerne som 1 vCPU kjerne.

Klusterarkitektur

For å kunne opprettholde høy tilgjengelighet, må kontrollplanet bestå av tre noder[20]. Dette betyr at hvis man ikke har tre kontrollplannoder som er operative, så er det ikke mulig å oppdatere klusteret. Dette er fordi klusteret trenger tilgang til tre *kluster-apiserver* komponenter.

Antall arbeidsnoder som trengs er avhengig av arbeidsmengde og applikasjoner som kjører på klusteret. Ved mye bruk av klusteret kan enten klusteret skaleres horisontalt ved å legge til en til arbeidsnode, eller vertikalt ved å bruke arbiendsnoder med tilgang til mer ressurser i form av CPU, RAM og lagring. Arbeidsnoder kan enkelt skaleres opp og ned ved behov.

[20] https://docs.okd.io/latest/architecture/control-plane.html#defining-masters_control-plane

4.4.4 Støttetjenester

Som nevnt tidligere, ved en UPI installasjon, må ulike støttetjenester settes opp før OKD klusteret kan settes opp. Denne seksjonen går i gjennom de ulike komponentene som utgjør støttetjenester. Disse tjenestene er

- DHCP for tildeling av IP-addresser.
- DNS for oppslag av tjenester.
- Lastbalanserer for dirigering av trafikk.
- PXE for å gi nodene tilgang til operativsystemet.
- Webserver for å dele ignition filer.

Alle disse tjenesten kjører på samme node kalt Service i figur 4.5.
Støttetjenestene er utenfor omfanget for hva OKD dekker i en UPI installasjon. Personen/gruppen som setter opp OKD klusteret er selv ansvarlig for valg av teknologi brukt for støttetjenester. For hver teknologi vil en kort redegjørelse være inkludert for hvilken programvare som er brukt for å levere tjenesten, og hvorfor denne er brukt.

DNS, lastbalansering og webserver er de eneste komponentene som er nødvendig, mens DHCP og PXE ikke er nødvending. Implementering av DHCP og PXE er likvel sterkt anbefalt da PXE automatiserer deler av prosessen og DHCP for å ha en sentral tjeneste som håndterer IP-adresser.

Videre i kapittel 4.4.5 beskrives hvordan disse tjenstene blir benyttet for å gjøre klar nodene før bootstrap prosessen installerer klusteret.

DHCP

DHCP er en tjeneste som vanligvis automatisk tildeler IP-adresser til noder. I dette scnearioet brukes DHCP til å statisk dele ut IP-adresser basert på MAC adresser. Når en ny node skal legges inn i klusteret blir MAC adressen til noden manuelt lagt inn i DHCP tjenesten. Når noden starter opp får noden en IP-adresse fra DHCP tjenesten basert på hvilken MAC adresse den har. MAC adresse er en unik identifikator til nodens nettverkskort. For å levere denne funksjonaliteten, har DHCP tjeneren *DHCPD* blitt benyttet. DHCPD, også kjent som ISC DHCP, er en av de første og mest kjente DHCP tjenerene for *nix systemer [21].

For å se konfigurasjonen som er benyttet i denne utrulleringen, se vedlegg D.1.3

DNS

Det finnes mange ulike programmer som kan levere en DNS tjeneste. DNS tjenesten som skal brukes i OKD klusteret trenger funksjonalitet for å

[21] https://en.wikipedia.org/w/index.php?title=DHCPD&oldid=990267504

kunne opprette soner, A/AAAA-, PTR og SRV records. Den offisielle dokumentasjonen til OKD har konfigurasjonseksempler basert på DNS tjenesten BIND[22], også kjent som *named*. Dette er en kjent og mye brukt DNS tjener innen Unix/Linux systemer som også vil bli benyttet i denne konfigurasjonen.

Klusteret er avhengig av A/AAAA for *name resolution* og PTR records for *reverse name resolution*. PTR records benyttes av FCOS til å navngi klusternodene, og for å generere forespørsler for sertifikat signering (CSR). For at en node skal kunne delta i klusteret, så må den ha et gyldig sertifikat som må aksepteres av en system administrator i klusteret.

Et SRV record[23] benyttes for å referere til tjenesten *_etcd-server-ssl*[24]. Denne spesifisere Time to live (TTL), prioritet, vekt, port og node. For hver kontrollplannode, så trenger man en SRV record. Denne recorden trenger også en A record som settes til samme IP-adresse som kontrollplannodene.

Det må opprettes konfigurasjonfiler for named som er filer for zone- og DNS records. *Zone* filene refereres til med en *include* i named.conf. I *zone* filen definerer to forskjellige soner, en for A/AAAA records og en for PTR records, og lokasjon for disse.
A/AAAA records er skrevet i formatet *<komponent>.<kluster navn>.<domene navn>.*, der komponent referer til en node eller en pod som opererer i OKD.

Eksempel på A og PTR records format:

A/AAAA:	bootstrap.lab.okd.local.	IN	A	10.0.10.11
PTR:	11	IN	PTR	bootstrap.lab.okd.local.

Tabell 4.2: Eksempel format for A- og PTR records.

[22] https://docs.okd.io/latest/installing/installing_bare_metal/installing-bare-metal.html#installation-dns-user-infra_installing-bare-metal

[23] https://en.wikipedia.org/w/index.php?title=SRV_record&oldid=984656612

[24] https://github.com/openshift/installer/blob/master/docs/user/metal/install_upi.md#dns-requirements

Oppsett av DNS records

Komponent	Record	Beskrivelse
Kubernetes API	api.<kluster navn>.<domene navn>	Identifiserer lastbalansereren for kontrollplannoder. Man må sette opp både A/AAAA eller CNAME og PTR records. Disse må kunne bli slått opp i DNS tjenesten for både klienter eksternt og for noder internt i klusteret.
	api-int.<kluster navn>.<domene navn>	Identifiserer lastbalansereren for kontrollplannoder. Det må settes opp både A/AAAA eller CNAME og PTR records. Disse må kunne slås opp i DNS tjenesten for noder internt i klusteret.
Ruter	*.apps.<kluster navn>.<domene navn>	Wildcard som refererer til lastbalansereren som sender trafikk til Ingress router pods. Som standard er dette arbeidernodene. Disse må kunne resolves for både klienter eksternt og for noder internt i klusteret.
Bootstrap	bootstrap.<kluster navn>.<domene navn>	Refererer til Bootstrap maskin. Man må sette opp både A/AAAA eller CNAME og PTR records. Disse må kunne reolves for noder internt i klusteret.
Kontrollplannoder	<master><n>.<kluster navn>.<domene navn>	Refererer til kontrollplannoder. Man må sette opp både A/AAAA eller CNAME og PTR records. Disse må kunne reolves for noder internt i klusteret.
Arbeidsnoder	<worker><n>.<kluster navn>.<domene navn>	Refererer til arbiedsnoder. Man må sette opp både A/AAAA eller CNAME og PTR records. Disse må kunne reolves for noder internt i klusteret.

Tabell 4.3: Konfigurasjon av DNS records.

For å se DNS konfigurasjons benyttet i denne installasjonen se vedlegg D.1.2.

Lastbalanserer

Det er behov for to typer lastbalansering. En API lastbalanserer som fungerer som et endepunkt for noder, og brukere for å interaktere og konfigurere klustert. Denne må støtte lag fire lastbalansering, og en tilstandsløs lastbalanseringsalgoritme. Den andre lastbalansereren er en applikasjons ingress lastbalanserer. Denne må også støtte lag fire lastbalansering og det anbefales at den er basert på vedvarnede tilkoblinger eller sesjoner.
Det finnes flere programvarer for lastbalansering som oppfyller disse kravene og mye mer. Gruppen har i tidligere prosjekter benyttet seg av HAProxy som lastbalanserer, og siden den oppfyller alle krav og gruppen er godt kjent med

verktøyet blir denne benyttet.

For API lastbalanserer må følgende porter åpnes:

Port	Backend noder	Intern	Ekstern	Beskrivelse
6443	Bootstrap, kontrollplan	X	X	Kubernetes API server
22623	Bootstrap, kontrollplan	X		Konfigurasjons server for noder

Tabell 4.4: Åpne porter for API lastbalanserer.

For applikasjons ingress lastbalansereren må følgende porter åpnes:

Port	Backend noder	Intern	Ekstern	Beskrivelse
443	Arbeidsnode	X	X	HTTPS trafikk
80	Arbeidsnode	X	X	HTTP trafikk

Tabell 4.5: Åpne porter for ingress lastbalanserer.

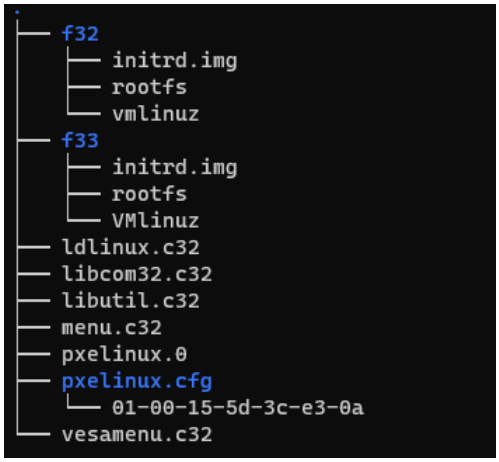
Ut i fra disse kraven, benyttes følgende konfigurasjon, vedlegg D.1.1.

Preboot Execution Environment

Når FCOS skal installeres på klusternodene, så oppstår det en utfordring. Som beskrevet i kapittel 4.4.3, så passer ikke golden image metoden spesielt bra for FCOS på grunn av sertifikater som kun er gyldig i 24 timer, og klusterspesifikk konfigurasjon i ignition filene. En mulig løsning på dette er å laste ned iso filen, tildele den til noden, starte opp noden og gi den informasjonen den trenger for å installere operativsystemet på ønsket måte. Dette fungerer greit hvis man skal sette opp en eller to noder, men hvis man skal sette opp mange noder eller teste utrullering av en infrastruktur er dette upraktisk. For å komme rundt denne utfordringen, uten lisenser, benyttes PXE boot.

Ved PXE boot henter noden alt den trenger for å installere operativsystemet over det lokale nettverket. Når en node etterspør IP-konfigurasjon ved oppstart, så får den også vite om *next-server* og en fil. *Next-server* er IP-adressen til Trivial File Transport Protocol (TFTP) server, som har en *boot-loader* fil med navnet *pxelinux.0*. Hvilken fil man refererer til er avhengig av om det er UEFI- eller BIOS-basert, i dette oppsettet er BIOS-basert benyttet.
Før man kan benytte seg av PXE boot, så må man konfigurere filer som forteller noden om hvor operativsystemet befinner seg, hvilken disk den skal installere på, og hvor den finner ignition filen for konfigurasjon av operativsystemet. Siden FOCS trenger en bestemt ignition fil basert på node type, benyttes nodens MAC-adresse, dette er den samme MAC-adressen som benyttes for å få tildelt IP-adresse. Navnet på denne filen har formatet

01-xx-xx-xx-xx-xx-xx, hvor xx er MAC-adressen. Basert på nodetypen som etterspør informasjon fra TFTP serveren, er filen konfigurert til å hente riktig ignition fil. For eksempel på fil format, se vedlegg D.1.4. Om MAC-addressen har bokstaver i seg, så kan ikke disse være store bokstaver. I tillegg til dette, så må man laste ned en kernel-, initramfs- og rootfs-filer.



Figur 4.7: Filstruktur for PXE miljø.

Figur 4.7 viser filstrukturen til mappen nodene henter oppstartsinformasjon fra. f33/32 inneholder filene for kernelen (VMlinuz), initi ramdisk (initrd.img) og filsystemet (rootfs). I mappen *pxelinux.cfg* ligger filene med navn som er satt til en nodes MAC-addresse[25].

Web server

Web serveren benyttes kun for å dele ignition filene for installasjon av FCOS på kluster nodene. På grunn av dette benyttes en enkel Apache web server, HTTPD, som konfigureres til å lytte på port 8080 i stedet for port 80 som er standard. Dette siden lastbalansereren er satt til å lytte på port 80 og begge tjenestene kjører på samme maskin som gjør at de ikke kan lytte på samme port. Etter at installasjonsprosessen er over, kan webserveren fjernes da det ikke lenger er bruk for den.

4.4.5 Utrullering av OKD

Etter at alle støttetjenestene er installert og konfigurert, så er det tid for untrullering av OKD klusteret. Dette starter med å lage en *install-config.yaml* fil. Denne filen inneholder blant annet informasjon som kluster- og domene navn, hvor mange kontrollplannoder og arbeidsnoder som settes opp, og hvilken ssh nøkkel som kan benyttes for å logge inn på nodene hvis problemer

[25] https://docs.oracle.com/en/operating-systems/oracle-linux/6/install/ol-pxe-boot.html

skulle oppstå vedlegg D.1.5. Når denne filen er konfigurert, benyttes OKD sitt installasjonsprogram, *openshift-installer*, for å først generere manifestfiler for Kubernetes, og deretter ignition filer for FCOS. Når man benytter installasjonsprogrammet for å generere manifestfilene, så konsumeres *install-config.yaml* filen, så det kan være lurt å ta en kopi før man starter om det er behov for denne i etterkant. Når ignition filene er generert, så kopieres disse til web serveren.

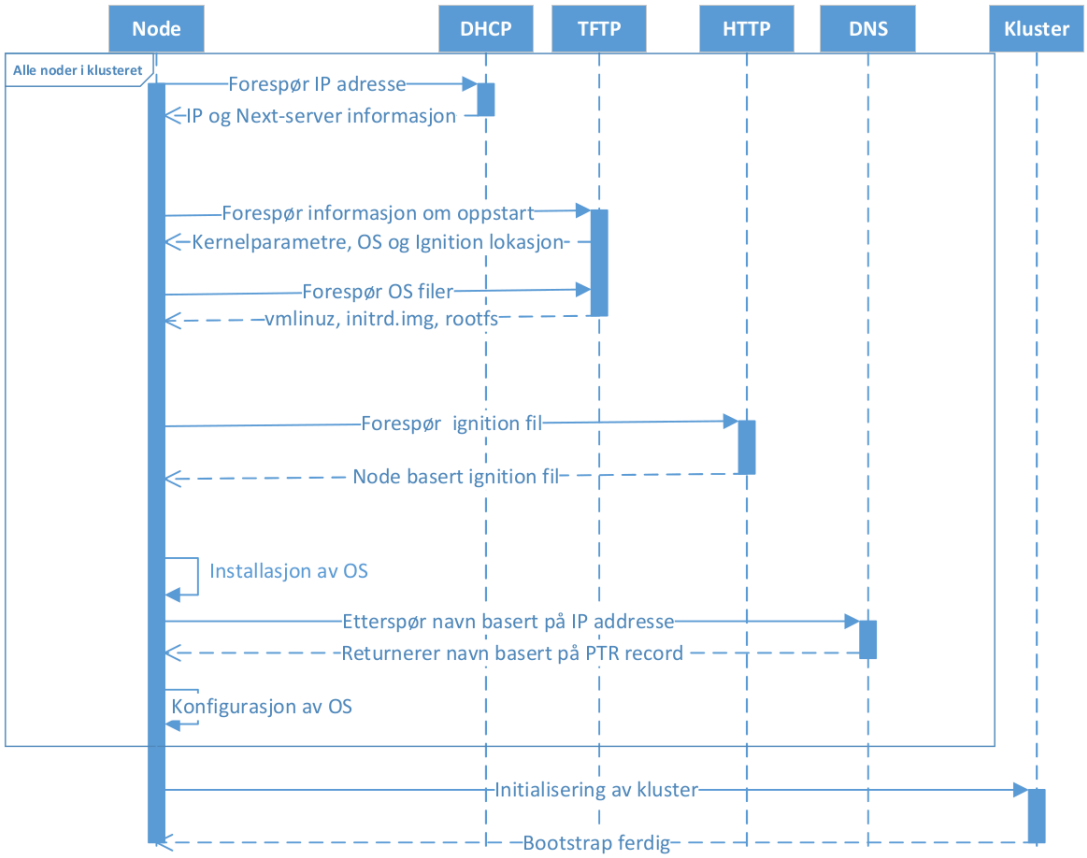
Etter denne prosedyren skal alt være klart for å starte bootstrap prosessen, og man kan starte bootstrap og kontrollplannodene. Når klusterinitialiseringen har startet så kan man starte arbeidsnodene.

Under testing av utrulleringen, er det observert at om man starter Bootstrap noden først, og venter til den er klar til å starte bootstrap prosessen før man starter kontrollplannodene og i tillegg bare starter en kontrollplannode, så er suksessraten for at utrulleringer er velllykket høyere enn ved oppstart av alle tre kontrollplannodene med en gang. De to andre kontrollplannodene kan startes etter at den første kontrollplannoden er ferdig oppsatt.
Nodene sender først en forespørsel om IP-addresse til DHCP tjenesten. Fra DHCP tjenesten får den tilsent en IP-addrese basert på MAC-addressen til noden, og en addresse til TFTP tjenesten som den skal kontakte videre. Noden kontakter deretter TFTP tjenesten angående informasjon den trenger for å starte opp, og får beskjed hvor operativsystemfilene kan hentes, hvilken kernel parametre den skal starte med og hvor ignition filen som benyttes for å konfigurere operativsystemet befinner seg. Når nodene har mottatt denne informasjonen etterspør den hvor operativsystemfilene fra TFTP tjenesten ligger og laster de ned. I neste steg hentes ignition filen fra web serveren notert som HTTP i figuren, Etter dette så installeres operativsystemet på noden. Når dette er gjennomfør og noden har starter på nytt, så kontatkter noden DNS tjenesten for å hente navnet noden skal ha. DNS tjenesten sender et svar basert på PTR records og deretter kan konfigurasjonen av operatiivsystemet starte. Etter denne prosessen er ferdig blir noden med i initialiseringen av klusteret som resulterer til slutt med at selve boostrap prosessen er ferdig i figur 4.6.

4.5 Konfigurasjon av OKD

I denne seksjonen vil grunnleggende konfigurasjon av OKD klusteret etter fulført installasjon beskrives. Dette inkluderer hvordan man legger til nye noder, opprettelse av brukere, og permanent lagring. Det er kun de mest grunnleggende delene som er beskrevet i denne seksjonen, se den offisielle dokumentasjonen videre konfigurasjon av OKD[26].

[26] https://docs.okd.io/latest/post_installation_configuration/



Figur 4.8: Sekvens diagram som viser stegene noder går i gjennom fra oppstart av noden, frem til initialisering av klusteret starter.

4.5.1 Brukerhåndtering

OKD opererer med to typer brukere. Den første typen bruker, tildeles til utviklere og administratorer som kan tildeles ulike rettigheter basert på roller definert i rolle-basert aksesskontroll (RBAC) som er beskrevet i kapittel 5.1. Den andre typen bruker er en *service bruker*. Dette er en bruker som tillater tilgang til API'er direkte, uten å dele en bruker sine innloggings detaljer. Denne brukeren tilhører et prosjekt, og kan tildeles ulike rettigheter på samme måte som den første typen bruker.

Autentisering av brukere

En autentiseringsmetode må opprettes før man kan opprette nye brukere. OKD støtter flere forskjellige metoder blant annet via gitHub/lab, google og LDAP[27]. For å vise hvordan man kan opprette nye brukere og gi dem tilgang

[27] https://docs.okd.io/latest/authentication/understanding-identity-provider.html#supported-identity-providers

til resusser, så benyttes et enkelt HTPasswd oppsett. HTPasswd benyttes kun for demonstrering, og er ikke nødvendigvis den beste løsningen for et produksjonsmiljø med tanke på sikkerhet og skalerbarhet.

HTPasswd

Uavhengig av hvilken innloggingsmetode som brukes, må de samme tre komponentene opprettes for å kunne autentisere seg til OKD klusteret. For å tilrettelegge for brukerautentisering med hjelp av HTPasswd, så trenger man en .htpasswd fil som genereres med hjelp av programvaren htpasswd , et OKD *secret* objekt og en *identity provider*. En *secret* er et objekt som kubernetes benytter for å lagre sensitiv data som passord, ssh-nøkkler og OAuth tokens. For å kunne benytte seg av en autentiserings løsning, så må en *identity provider* opprettes. En identity provider oppretter en kobling mellom brukernavn og passord i htpasswd filen, og secreten. Dette legges så inn i *OAuth* konfigurasjon.

Brukertilganger

Etter at man har opprettet en ny *idenetity provider* og en ny administrator bruker med cluster-wide rettigheter, så anbefales det å fjerne den midlertidige *kubeadmin* brukeren vedlegg E.1. Dette vil øke den overordnede sikkerheten i klusteret. Husk å opprettet en ny administrator bruker og undersøk at den har rette tilgangene før man fjerner kubeadmin brukeren, hvis ikke mister man administrator tilgang til klusteret.

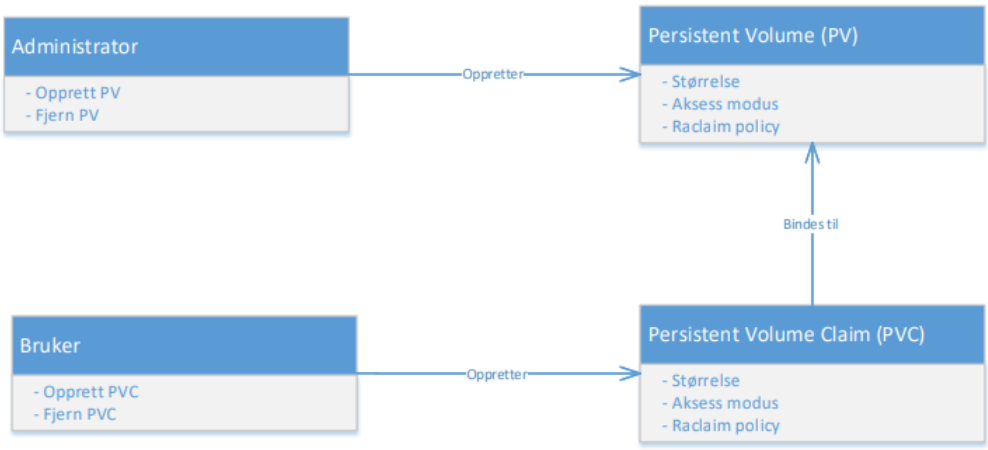
4.5.2 Lagring

Det tas ikke høyde for hvilken underliggende lagringsløsning som benyttes når det settet opp permanent lagring for OKD. Dette betyr at man kan benytte ulike lagringsløsinger som for eksempel et distribuert filsystem eller en vanlig disk.

OKD benytter Kubernetes sitt *persistent volum (PV)* rammeverk. Dette tilrettelegger for provisonering av permanent lagring av en administrator. Når et PV opprettes så spesifiseres blant annet størrelse, aksess modus og *reclaim policy*. Reclaim policy bestemmer hva som skal skje med et PV når den ikke lengre er bundet til et *persistent volume claim (PVC)*. Når en bruker ønsker å benytte seg av et PV, gjøres dette i form av en PVC. Ved opprettelse av en PVC så spesifiseres blant annet størrelse og aksess modus. Hvis et PV med tilnærmede like kriterier finnes, så bindes PVCen til denne. Når bindingen er oppstått, så kan ikke en ny PVC bindes til den samme PVen.
Når en bruker ikke lengere har behov for PVCen og den slettes, så vil PV som den var bundet til vil få *released* status og evt. data håndteres i forhold til *reclaim policy*. Det er to reclaim policyer som er tilgjengelig, retain og delete.

Retain tillater manuell prosess for gjenvinning av PVen, mens delete sletter PVen etter bruk. Det finnes en tredje policy recycle, men denne er ikke lengre støttet.

Man kan også benytte dynamisk provisonering av permanent lagring. I dette tilfellet så opprettens et nytt PV når en bruker oppretter en PVC.



Figur 4.9: Sammenheng mellom PV som er opprettet av en administrator og en PVC laget av en bruker.

For å se eksempler på hvordan dynamisk lagring kan konfigureres, se vedlegg D.2.2.

Image registry

For at man kan begynne å utrullere applikasjoner til klusteret, så trengs det en plass å lagre konteiner imagene som benyttes. Dette gjøres ved å opprette et lokalt image registry. Et image registry brukes av images som bygges på klusteret, og som kilde for images som benyttes i jobber på klusteret. Når et nytt image legges til i registriet, så får klusteret beskjed og reagerer til endringene i det nye imaget.
Et image registry opprettes på samme metoden som et vanlig PV, men man må gjøre noen endringer i kluster operatoren for *image-registry*, og man må ha en minimums kapasitet på 100 GB. For å se kommandoer for dette, se vedlegg E.1, og vedlegg D.2.1 for konfigurasjons filer.

4.5.3 Legg til nye noder

For å skalere klusteret opp ved å legge til flere kontrollplan eller arbeidsnoder, så benyttes samme installasjons metode som ved kluster installasjon, men uten

bootstrap noden. Før man kan starte denne prosessen, så må ignition filene oppdateres med et gyldig sertifikat. For å se kommando for å hente ut sertifikat og oppdatere ignition filen, se vedlegg E.1. I tillegg til dette, så må DHCP, DNS, lastbalansering, PXE oppdateres for å tilrettelegge for den nye noden. Når filen er oppdatert, kopiert til webserveren og støttetjenestene er oppdatert, så kan den nye noden startes.
Under installasjonen av den nye noden, så vil den sende inn en forespørsel for signering av sertifikatet den har. Denne forespørselen må godtas før noden blir en del av klusteret. For kommando se vedlegg E.1.

Kapittel 5

Sikkerhet

OKD er en ferdig utstyrt distribusjon av Kubernetes med et fokus på sikkerhet. Kapitlet tar for seg rolle-basert aksesskontroll, det forklarer grunnleggende om konteinersikkerhet og hvilke sikkerhetsutfordringer OKD løser ved bruk av rootless konteinere.

5.1 Rolle-basert aksesskontroll

OKD bruker RBAC for håndtering av tilgangskontroll til ulike ressurser i klusteret. RBAC er en komponent fra kubernets og benytter regler, roller og bindinger for å håndheve tilganger.

Det som skiller OKD sin implementasjon av RBAC over standard kubernetes, er at den kommer med flere forhåndsdefinerte roller. RBAC håndteres ved å benytte tre ulike komponenter. Den minste komponenten i RBAC er en regel. En regel er et nøkkelord som tillater en handling. Dette kan for eksempel være om en bruker har mulighet til å liste alle poder i et prosjekt. For å gjøre tildelingen av regler enklere, så kan flere regler samles til en rolle. Den siste komponenten er en binding som kobler en rolle eller regler til en bruker.

Både roller og bindinger har ulikt omfang ut i fra hvilken sammenheng de benyttes i. De kan være *cluster-wide* hvor de påvirker hele klusteret, eller *local* hvor kun et prosjekt påvirkes. Når disse benyttes er det noen regler som gjelder. En cluster-wide binding kan kun benytte cluster-wide roller, mens en local binding kan benytte både local roller og cluster-wide roller. Et eksempel på dette er å benytte en local binding på cluster-wide rollen *cluster-admin*. Dette vil gi en bruker cluster-admin tilgang for et prosjekt men ikke for hele klusteret.

Ved å dele roller og bindinger opp i disse to ulike nivåene, kan roller gjenbrukes i ulike sammenhenger.

Rolle	Beskrivelse
admin	Prosjekt manager. Hvis den benyttes i en lokal binding, så kan brukeren se og endre alle resurser i prosjektet utenom kvoter.
basic-user	Kan kun få grunnleggende informasjon om prosjekter og brukere.
cluster-admin	En *super-user* som har tilgang til all funksjonalitet for alle prosjekter. Hvis man binder denne rollen til en lokal binding, så vil brukeren ha samme tilganger som *admin* rollen, men også kontroll over kvoter.
cluster-status	Tillater en bruker å se grunnleggende kluster status informasjon
edit	Kan endre de fleste objekter i et prosjekt, men kan ikke se eller endre roller eller bindinger.
self-provisioner	Kan opprette egne prosjekt.
view	Kan ikke gjøre noen endringer, men kan se de fleste objektene i et prosjekt. Kan ikke se eller endre roller eller bindinger.

Tabell 5.1: Oversikt over roller som kommer med OKD. Hentet fra OKD dokumentasjon[1].

5.1.1 Prosjekter

For å kunne tilrettelegge for flere brukere eller grupper av brukere i klusteret så benytter OKD Prosjekter. Dette er det samme som namespaces i Kubernetes. Ved å benytte ulike prosjekter, så hindrer man grunnleggende navnekolisjoner, tilrettelegger for autorisasjons mekanismer for deler av klusteret, begrense ressursutnyttelse for brukere eller grupper og isolasjon mellom ulike brukere og grupper. Alle prosjekter har sine egne objekter, regler, begrensninger og service brukere.

5.2 Konteinersikkerhet

Denne seksjonen går i gjennom de grunnleggende prinsippene bak en konteiner. En konteiner er en prosess som kjører på en host maskin, med et begrenset syn til host maskinen og kun tilgang til en del av filsystemet. Siden det er en prosess som kjører på hostens operativsystem, så deler den kernelen med hosten[16].

For å gjøre en prosess til en konteiner så isolerer man prosessen ved hjelp av namespaces, endrer hva prosessen tror er roten til filsystemet og begrenser resursser den kan benytte via kontroll grupper (cgroups)[17]. Denne isoleringen

[1] https://docs.okd.io/latest/authentication/using-rbac.html#default-roles_using-rbac

er sett i fra konteinerens siden og host maskinen kan fremdeles se hva en konteineren gjør[18]. Et namespace er en metode å gruppere globale system resursser til en abstraksjon som en prosess, i det samme namespacet, oppfatter som en isolert instans av resurssen[2]. En endringer i et namespace vil kun påvirke prosesser i det namespacet og ikke er synlig for prosesser utenfor. I Linux kernel 5.6 er det åtte ulike namespaces som støttes[3].

- Unix Timesharing System (UTS)
- Prosess ID (PID)
- Mount punkter (mount)
- Nettverk (network)
- Bruker og gruppe IDer (user)
- Inter-prosess komunikasjon (IPC)
- Kontroll grupper (cgroups)
- Tid (time)

Cgroups benyttes for å begrense mengden resursser en konteiner har tilgang til. Dette kan være blant annet CPU og minne en eller flere prosesser kan utnytte. Dette kan hindre at en prosess utnytter så mye resursser at det har en negativ effekt på andre prosesser. Siden man isolerer en prosess fra andre prosesser ved hjelp av namespaces, så er det naturlig å isolere filsystemet prosessen har tilgang til også. Dette kan oppnås ved å endre hva prosessen tror er roten i filsystemet. Ved å endre roten så hindrer man prosessen i å se høyere opp i filsystem treet til host maskinen enn hva den nye roten er satt til.
Når man endrer roten til filsystemet så er det viktig å huske at prosessen kun har tilgang til de filen/programmene som befinner seg i den nye filsystemet. Dette betyr at hvis man oppretter en ny rot uten innhold, så kan ikke prosessen utføre noen oppgaver. Når man kjører en konteiner så er det vanlig å basere den på et image som er konfigurert til å gjøre en type oppgave. Dette kan for eksempel være en web server. Et image er basert på en Linux distribusjon, og alle nødvendige programmer/filer er tilgjengelig for oppgaven den skal utføre, og filer/programmer som ikke er nødvendig fjernes. Når man benytter en konteiner så har den et spesifikt formål. For å øke sikkerheten enda mere, så kan konteineren plasseres i en *sandbox*. En type sandbox som benyttes i operativsystemet til kluster nodene, FCOS, er Security Enhanced Linux.

5.2.1 Security Enhanced Linux

Security Enhanced Linux (SELinux) er en Linux sikkerhets modul (LSM) som har sine røtter fra United States National Security Agency (NSA), og er utviklet av Red Hat[4].
Oppgaven til SELinux er å begrense hva en prosess kan gjøre når det kommer til interaksjon med filer eller andre prosesser ved hjelp av en *policyer*. Måten

[2]https://en.wikipedia.org/w/index.php?title=Linux_namespaces&oldid=1019273454
[3]https://www.man7.org/linux/man-pages/man7/namespaces.7.html
[4]https://www.redhat.com/en/topics/linux/what-is-selinux

dette gjøres på er at alle prosesser er en del av et *SELinux domene*, og alle filer har en *type*. SELinux benytter ikke bruker identitet for tilgangs kontroll, den benytter *lables*. Under SELinux så tildeles alle filene på maskinen et label. Denne lablen benyttes av SELinux når en *policy* skal håndheves. Det er policyen som bestemmer hva en prosess i et bestemt SELinux domene kan utføre på en fil med et lable. Operativsystemer som benytter SELinux er eksempler på et *Mandatory Access Controll* (MAC) system. Dette betyr at når en administrator oppretter en policy for en fil så kan ikke en bruker, intensjonelt eller ved uhell, endre tilgangen til den filen selv om brukeren er eieren. Dette tillater administratorer og definere sentrale policyer som må håndheves av alle brukere. SELinux opererer sammen med tilgangskontrollen i Linux, også referert til som *discretionaty access control* (DAC). Hvis man ikke har tilgang til en fil p.g.a. DAC, så får man ikke tilgang til filen via en SELinux policy. Begge tillatelsene må tilfredsstilles for å få tilgang til filen.

5.3 Sikkerhetsutfordring med konteinere

En sikkerhetsutfordring med konteinere har vært at de kjøres av en priviligert bruker på host maskinen. Hvis konteineren blir kompromittert og aktøren bryter seg ut, så vil aktøren ha priviligert tilgang til host systemet. En metode for å hindre dette scenarioet, er å benytte *rootless* konteinere.

5.3.1 Rootless konteinere

I en rootless konteiner så benytter man seg av namespacet som omhandler brukere (User namespace). Her oppretter man en kobling mellom en ikke rot bruker på host maskinen til en bruker i konteineren. Denne brukeren kan være en vanlig bruker eller en rot bruker i konteineren. Brukeren i konteineren er medlem av rot gruppen for å kunne benytte seg av filer som eies av rot. Dette tillater brukeren å utføre handlinger som trenger rot tilgang i konteineren, men hvis den prøver å utføre en handling som trenger rot tilgang på host maskinen, som for eksempel å binde en port med et lavt port nummer, så blir handlingen hindret.

```
1  .
2  .
3  .
4  ARG java_image_tag=11-jre-slim
5  FROM openjdk:${java_image_tag}
6  ARG spark_uid=185
7  .
8  .
9  .
10 RUN set -ex && \
11     sed -i 's/http:\/\/deb.\(.*\)/https:\/\/deb.\1/g' /etc/apt/
       sources.list && \
12     apt-get update && \
13     ln -s /lib /lib64 && \
14     apt install -y bash tini libc6 libpam-modules krb5-user
       libnss3 procps && \
15     mkdir -p /opt/spark && \
16     mkdir -p /opt/spark/examples && \
17     mkdir -p /opt/spark/work-dir && \
18     touch /opt/spark/RELEASE && \
19     rm /bin/sh && \
20     ln -sv /bin/bash /bin/sh && \
21     echo "auth␣required␣pam_wheel.so␣use_uid" >> /etc/pam.d/su &&
       \
22     chgrp root /etc/passwd && chmod ug+rw /etc/passwd && \
23     rm -rf /var/cache/apt/*
24 .
25 .
26 .
27 # Specify the User that the actual main process will run as
28 USER ${spark_uid}
```

Kodeliste 5.1: Utdrag fra vedlegg C.2 av dockerfil for å bygge Apache Spark.

Dette er et utdrag fra vedlegg C.2 som er en av Dockerfilene som er brukt for å lage konteiner av Apache Spark. Dockerfilen installerer programmer som er nødvendige og utfører operasjoner som root fra linje 10 til linje 23. Etter dette er ferdig trenger man ikke lengre priviligeiene root brukeren har tilgang til og endrer derfor til en bruker som ikke har disse tilgangene, som i dette tilfellet er brukeren kalt 185. Dette gjøres i linje 28 der brukerid tas som et argument fra linje 6. Brukeren 185 er den som kjører Apache Spark prosessene og vil utgjøre en mindre sikkerhetsrisiko om den blir kompromittert.

OKD benytter rootless konteinere som standard. Alle prosjekter i OKD, er tildelt et sett med bruker id'er (UID), gruppe id'er (GID) og SELinux lables. Disse er unike og overlapper derfor ikke med noen andre prosjekter. Når man utrullerer en ny konteiner, så tildeles den en tilgjengelige UID og GID for prosjektet. UID som benyttes inne i konteineren bindes da til en bruker med samme UID på vertsmaskinen, som er eier av prosessen. Dette gjør at hvis en ondsinnet aktør kompromitterer en kontiener og bryter seg ut, så vil aktøren være en ikke priviligert bruker. Man kan fremdeles sette en rot bruker til å kjøre prosessen og som UID i konteineren, men dette krever spesielle tilganger.

Kapittel 6

Utprøving av dynamisk allokering på test og utviklingsinfrastruktur

I en vanlig arbeidsoppgave i Apache Spark spesifiseres et bestemt antall arbeidspoder som skal utføre dataanalysen. Antallet arbeidspoder må spesifisers på forhånd ut i fra hva brukeren som er ansvarlig for analysen tror jobben trenger av ressurser. Dette kan resultere i at jobben tar lengre tid siden siden det har for få arbeinderpoder, eller man benytter for mye ressurser fordi det er for mange arbeidspoder i forhold til det som trengs.

Apache Spark støtter dynamisk ressurs allokering som automatisk skalerer arbeidspoder basert på arbeidsmengde. Denne funksjonaliteten kan bidra med å få bedre ressursutnyttelse i OKD klusteret.
I dette kapittelet vil de ulike komponentene i infrastrukturen som er benyttet forklares, og deretter vil utprøvelse av dynamisk skalering av Apache Spark beskrives i test og utviklingsinfrastrutkuren.

6.0.1 Beskrivelse av arkitektur brukt for test og utvikling

I denne seksjonen vil en kort beskrivelse de ulike komponentene test og utviklingsmiljøet består av, og hvordan de henger sammen. Testmiljøet kombinerer mange ulike teknologier som er forklart i teori og design kapittel, og denne seksjonen vil være en konsept utprøvning på hvordan disse kan operere sammen.

OKD
For å utføre testing så benyttes et OKD kluster i NTNU sin SkyHigh infrastruktur. Denne er satt opp ved hjelp av en IPI installasjonsmetode[1] for OKD versjon 4.7.0. Klusteret er bygget opp av tre kontrollplannoder og tre arbeidsnoder. Kontrollplannodene har tilgang til 4 vCPU og 16GB ram, mens en arbeidsnode har 4 vCPU og 16 GB ram og de siste to

[1] https://docs.okd.io/latest/installing/installing_openstack/installing-openstack-installer-custom.html

har 2 vCPU og 8GB ram. Dette gir klusteret en total på 20 vCPU og 80GB ram.

HDFS
Som nevnt i kapittel om HDFS, så er HDFS satt opp som et pseudo distribuert system. Noden som er ansvarlig for HDFS har tilgang til en vCPU, 2 GB ram og 40GB lagringsplass. Med denne konfigurasjonen ble det ved flere anledningner observert ustabilitet og i noen tilfeller skrudde noden seg av. Løsningen på dette var å øke mengden ressurser noden hadde tilgang til. Ved å doble mengden CPU og ram noden har tilgjengelig, forsvant ustabiliteten. Størrelsen på noden brukt under testing er 2 vCPU, 4 GB ram og 40 GB med lagringsplass.

DNS
For at Apache Spark podene skal få informasjon om hvor den skal kontakte HDFS, er en enkel DNS server satt opp.

History server
For å kunne hente ut logg data settes det opp en Apache Spark history server. Denne henter logg data fra jobben som utføres fra HDFS og visualiserer dette i et grafisk grensesnitt. Som standard er informasjon om en Spark applikasjon kun tilgjengelig under runtime uten bruk av Apache Spark history server[2].

Web server
Enkel web server som Apache Spark driverpoden henter spark jobben fra. Ved bruk av en web server kan jobber konfigureres og endres på kjapt for å gjøre små justeringer. I et produksjonsmiljø anbefales det at Apache Spark henter jobben fra HDFS.

GCP spark operator
For å starte opp analysejobben brukes GCP spark operator fremfor spark submit da den har mer mulighet for konfigurasjon og kan konfigureres som et Kubernetes objekt.

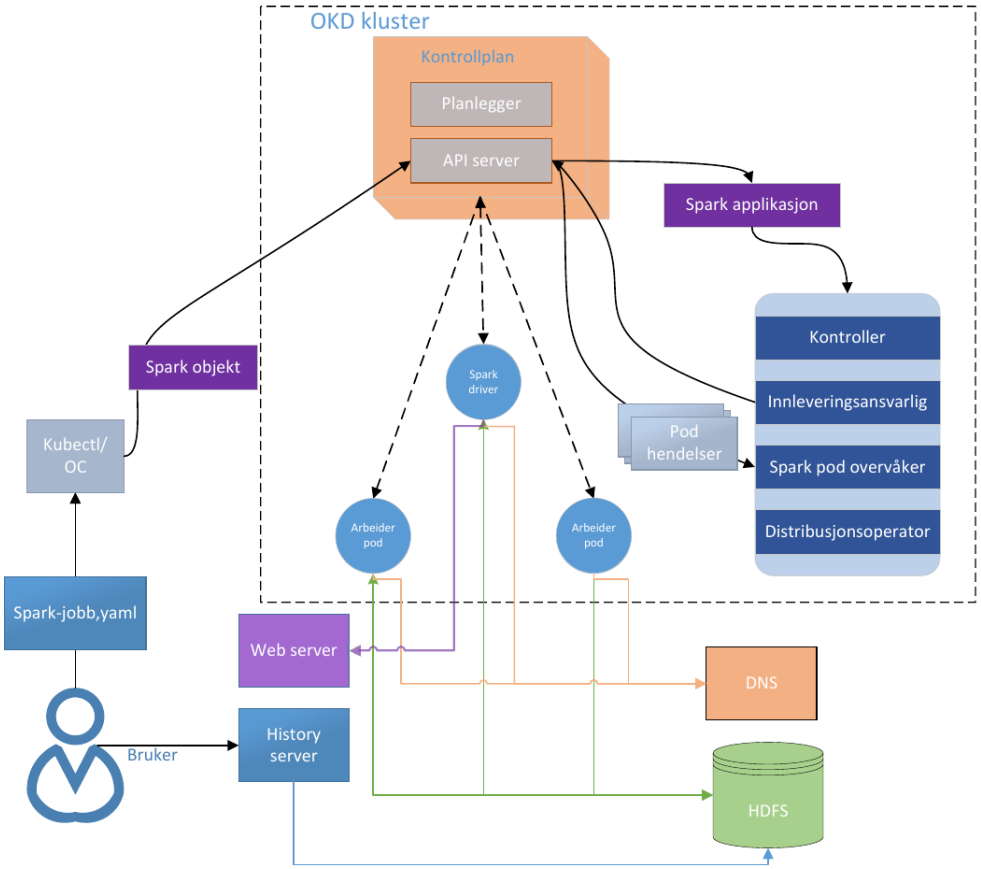
6.0.2 Arkitektur

Spark.jobb.yaml er en yaml fil som definererer hvordan Spark applikasjonen skal utføres. Hele filen De mest sentrale konfigurasjonsinnstilligene i denne filen er.

- Spesifiserer at den skal sende loggfiler til HDFS
- Informasjon om hvilken DNS den kan bruke for å gjøre oppslag for adressen til HDFS
- Informasjon om hvilken image den skal benytte
- Hvor spark jobben kan hentes

[2]https://spark.apache.org/docs/latest/monitoring.html

- Antall arbeiderpoder og CPU og minne som tildeles driver og arbeider pod.



Figur 6.1: Videreutvikling av figur 4.2 med alle tjenester satt opp i test og utviklingsinfrastrukturen.

Spark.jobb.yaml filen sendes som et Spark objekt ved bruk av Kubectl/OC til OKD sin Api server. API serveren tar deretter og sender spark objektet videre som en spark applikasjon til GCP. I GCP er et av hovedmomentene at DNS konfigurasjonen går i gjennom en mutating admission webhook som er administrert av GCP operator. I denne prosessen endres DNS konfigurasjonen til å være den som ble spesifisert i spark.jobb.yaml i steden for konfigurasjons-informasjon som OKD benytter[3]. Deretter utrulles spark driveren som henter spark jobben fra webserveren. Driverpoden starter så arbeiderpodene. Alle spark podene kontakter DNS serveren for opplysinger om hvor HDFS ligger. Deretter får arbeiderpodene tildelt jobben og lagrer loggfiler til HDFS når de er ferdige. Spark history server brukes som et grensesnitt for loggdata i HDFS der bruker kan interaktere med data for å få informasjon om jobben.

[3]https://kubernetes.io/blog/2019/03/21/a-guide-to-kubernetes-admission-controllers/

6.0.3 DNS konfigurasjon i Apache Spark konteiner

For å få Apache spark podene til å kunne nå HDFS for å hente data, trenger de DNS for å få informasjon om hvor HDFS er. En enkelt løsning for dette er å endre /etc/hosts filen i konteineren. Denne filen oversetter hostname og domenenavn til IP-adresser. Dette ble utført ved å legge inn følgende kommando i Dockerfilen som blir brukt for å bygge konteineren.

```
COPY host.txt /etc/hosts
```

Her er host.txt en tekstfil som inneholder IP-adressen til HDFS noden og navn.

```
10.212.143.27 testhdfs.novalocal
```

Da konteineren ble kjørt lokalt, ble /etc/hosts filen overskrevet og i teorien skulle dette også fungere i Kubernetes. Under testing av konteineren i Kubernetes fikk ikke Apache Spark tak i HDFS. For å feilsøke problemet ble en Python kommando lagt til i Spark jobben som printer ut innholdet i /etc/hosts filen.

```
with open('/etc/hosts', 'r') as f:
        print(f.read())
```

Ved å se på loggfilene fra Apache Spark podene inne i OKD så vi at /etc/hosts hadde blitt overskrevet av Kubernetes. Kubernetes administrerer denne filen selv for å styre DNS konfigurasjon til poder. Kubernetes har funksjonalitet for å endre DNS innstillinger i en pod og dette ble prøvd uten hell. Følgende ble lagt inn i Spark applikasjonen.

```
  dnsConfig:
      nameservers:
        - 10.0.3.155
      options:
        - name: ndots
          value: "1"
        - name: edns0
```

Etter å ha lest i dokumentasjonen til GCP Spark operator ble det oppdaget at det trengs en mutating admission webhook. En mutilating webhook kan endre pod konfigurasjon etter at den er sendt til APIen, og før den lagres og poden opprettes [4]. Denne funksjonaliteten implementeres i det GCP spark operatoren utrulleres til klusteret. se vedlegg E.1.

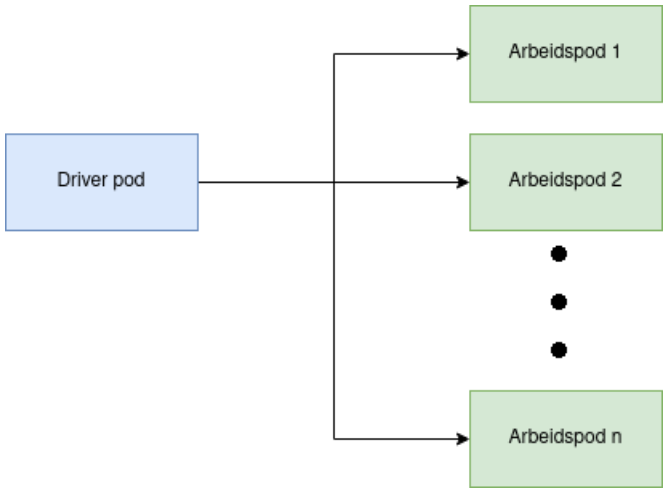
Etter at en mutating admission webhook var implementert i GCP spark operatoren fungerte DNS innstilligene som ble lagt inn i Spark applikasjonen og Apache spark podene fikk tilgang til HDFS ved å kontakte DNS serveren. En annen mulighet for å endre DNS konfigurasjon for poder, uten å ta i bruke mutilating webhooks, er ved å endre innstillinger til DNS operatoren i OKD klusteret.

[4] https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/blob/master/docs/quick-start-guide.mdabout-the-mutating-admission-webhook

6.1 Test av dynamisk skalering for en batch jobb

Formålet med testen er å se hvordan Apache Spark selv skalerer poder basert på arbeidsmengde i en spark applikasjon. Dette innebærer både å skalere opp når det kreves mer ressurser, og å skalere ned om den har mer ressurser enn den trenger.

Spark applikasjonen skalerer opp arbeiderpodene når den har oppgaver som ikke er delt ut til arbeidspoder enda. Dette foregår rundebasert og arbeiderpodene skaleres opp slik at det nye antallet arbeidspoder er det dobbelt av det forrige antall arbeidspoder. Det vi si at om det var to arbeidspoder opprinnelig, og Spark applikasjonen har udelegerte oppgaver, skaleres antaller opp til fire arbeidspoder [5]. For nedskalering av arbeidspoder skjer dette om en arbeidspod har vært inaktiv etter et gitt tidsintervall.



Figur 6.2: Dynamisk skalering av arbeidspoder. Driverpoden koordinerer og lager arbeidspodene

6.1.1 Generering av testdata

Testdata har data blitt generert fra å hente ut bokstaver og tall fra /dev/urandom for å lage en tekstfil på en million linjer. Majoriteten av linjene er unike. Formålet med data som genereres er ikke å lage testdata som kan kjøres på et annet system for å sammenligne resultater da data ikke er reproduserbart. Testdataen generert er kun brukt for å lage en arbeidsmengde som blir benyttet til å utprøve dynamisk allokering.

```
tr -dc "A-Za-z␣0-9" < /dev/urandom | fold  -w100 | head -n
    1000000 > file.txt
```

Kodeliste 6.1: Kommando for å generere testdata.

[5] https://spark.apache.org/docs/latest/job-scheduling.htmlresource-allocation-policy

Outputen fra denne kommandoen har blitt lagret til fil kalt *1m*. Deretter har innholdet i fil 1m blitt iterert over to, fem og ti ganger for å lage større filer kalt 1m, 1mx2, 1mx5 og 1mx10. I begynnelsen av testfasen ble det forsøkt å bruke testdata med 2 000 000, 5 000 000 og 10 000 000 unike linjer. For å kunne fullføre disse jobbene, hadde ikke podene tilgang til nok ram. Dette medførte at OKD klusteret stoppet podene. For å komme rundt denne utfordringen uten å ha tilgang til mere ressurser, ble det valgt å iterere over samme 1m filen for å øke linjeantallet med å ha duplikater av testdata.

6.1.2 Spark jobb

Jobben som benyttes for å teste dynamisk skalering er en lett modifsert versjon av eksempelapplikasjonen wordcount.py der applikasjonen er satt til å hente en tekstfil fra HDFS i stedet for å hente lokalt[6]. Programmet teller antall forekomster av hvert enkelt ord som er i den genererte filen.

```
import sys
from operator import add
from pyspark.sql import SparkSession

if __name__ == "__main__":
    spark = SparkSession\
        .builder\
        .appName("PythonWordCount")\
        .getOrCreate()

    line = spark.read.text("hdfs://dnshost.internt:9000/user/
       fedora/input/1m.txt")
    print(line)
    lines = line.rdd.map(lambda r: r[0])
    counts = lines.flatMap(lambda x: x.split(' ')) \
                  .map(lambda x: (x, 1)) \
                  .reduceByKey(add)
    output = counts.collect()
    for (word, count) in output:
        print("%s: %i" % (word, count))
    spark.stop()
```

Kodeliste 6.2: Apache Spark jobb i Python brukt i testing.

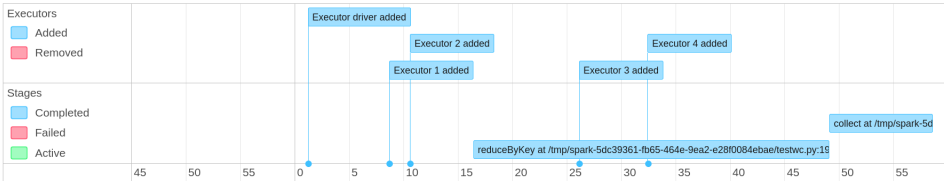
6.1.3 Skaleringstest

Under utføring av test ble det oppdager problemer med at Apache Spark podene ikke fikk tak i noden som kjører HDFS og DNS i infrastrukturen.
I noen tilfeller fungerte alt som det skulle og Spark applikasjonen var vellykket, mens i andre tilfeller ble det observert problemer med DNS. Feilmeldigen er følgende: *java.net.UnknownHostException: dnshost.internt* En interessant observasjon er at ved større antall Apache Spark arbeiderpoder som kjørte var

[6] https://github.com/apache/spark/blob/master/examples/src/main/python/wordcount.py

det høyere sansynlighet for DNS problemer. Men det har også hvert problemer når en eller to arbeidspoder har kjørt. Dette har resultert i at den innsamlede datamengden er mangelfull og at siden det har oppstått problemer som ikke nødvendigvis skyldes Apache Spark kan ikke testen konkludere om dette er en god løsning eller ikke. Videre beskrives hva som er observert fra noen av testene.

Oppskalering



Figur 6.3: Bilde hentet fra Apache Spark history server. Spark applikasjonen starter med to arbeiderpoder, og skalerer opp til fire.

Testen ble utført med at Apache Spark applikasjonen startet med to arbeiderpoder, med mulighet til å skalere opp til fire. Dette ble utført på filen 1m, 1mx2 og 1mx5 fra kapittel 6.1.1. På filen 1mx10 måtte den ha tilgang til fire arbeiderpoder hvis ikke fulførtes ikke jobben på grunn av problemer med DNS. For de andre jobbene ble det observert at spark applikasjonen startet opp nye arbeiderpoder basert på størrelsen på jobben.

Nedskalering

For å teste nedskalering av Apache Spark, var tanken å benytte for et høyt antall arbeiderpoder i forhold til jobben som krevdes. På grunn av ressursbegrensninger i SkyHiGh, var det ikke mulighet å lage flere enn fem arbeiderpoder. Dette førte til at vi ikke fikk laget et falsk høyt antall arbeidepoder i forhold til arbeidsmengden som skulle utføres og det ble ikke mulig å observere at automatisk nedskalering fungerer. Grunnen til dette er at ved å benytte en liten arbeidsmengde og bruke arbeidspoder med mindre ressurser, fullføres jobben for kjapt til at nedskalering tar effekt.

Kapittel 7

Avslutning

7.0.1 Konklusjon

Denne oppgaven dekker en grunnleggende bare-metal installasjon og konfigurasjon for vSphere. Apache Spark er blitt konteinerisert og testet på den oppsatte OKD platformen i SkyHiGh. Det er utført analysejobber for å teste automatisk skalering i Apache Spark og kompatibilitet med det distribuerte filsystemet HDFS. I tillegg til dette, er det opprettet dokumentasjon som viser hvilke kommandoer som er benyttet samt konfigurasjonseksempler for de ulike tjenestene.
I begynnelsen av prosjektet ble det i gjennom diskusjon formulert kriterier som må oppfylles før systemet kan settes ut i produksjon. I denne seksjonen blir produktet som er utviklet i prosjektperioden sammenlignet med kriteriene satt i kapittel 2.2.1.

Sikkerhet

Den generelle sikkerheten i systemet er på et høyt nivå. OKD har en rekke sikkerhetsfunksjonalitet som RBAC og rootless konteinere som bidrar med å øke sikkerhetsnivået. Sikkerhet for OKD er godt dekket i rapporten, men på grunn av mangel på tid er ikke sikkerhet i HDFS og Apache Spark gått i gjennom. Dette innebærer hovedsaklig autentisering mellom HDFS og Apache Spark.

Skalerbarhet
OKD er installert som en UPI bare metal installasjon der mye av prosessene for å legge til og fjerne noder krever manuelt arbeid. UPI bare metal er skalerbar, men dette tar tid. En IPI installasjon skalerer bedre enn måten det er løst i denne rapporten, siden IPI støtter automatisk skalering av noder. IPI installasjon er ikke et alternativ på nåværende tindspunkt for NTNU SOC, som innebærer at skalering er mulig, men er en manuell prosess.
Apache Spark er skalerbart og kan takle varierende arbeidsmengder, men som

standard må det spesifiseres ressurser som skal bli benyttet på forhånd. I kapittel 6 ble det gjort forsøk med automatisk skalering der Apache Spark selv skalerer Poder basert på arbeidmengde. Dette fungerte ikke som tiltenkt i vårt tilfelle grunnet problemer med DNS.

Dokumentert
Fra desember i fjor til mai i år har det vært en markant forbedring i dokumentasjonen for OKD. Flere installasjonsmetoder har kommer i løpet av perioden som kunne vært alternativ for den gruppen har valgt. OKD er i dag greit dokumentert, men om den utviklingen vi har sett i løpet av prosjektperioden fortsetter, så vil dette være godt dokumentert i fremtiden. Gruppen har skrevet dokumentasjon for arbeid utført i rapporten og lagt ved relevante konfigurasjonseksempler som vedlegg som kan brukes for å sette opp et lignende system.

Pålitelighet
Basert på vår erfaring, er OKD trøblete å sette opp. Både IPI installasjon brukt i testing og utvklingsinfrastrukturen, og UPI for produksjonsinfrastrukturen har blitt installert flere ganger, og selv om installasjonsprosessen er gjennomført ved bruk av lik metode kan utfallet av installasjonsprosessen variere fra gang til gang. OKD er upålitelig å sette opp, og har noen ganger fungert i et par dager før det har gått ned. Dette skyldes hovedsaklig autentisering og API komponenter som har gått ned. Den siste utrulleringen av OKD ved bruk av IPI i test og utvklingsinfrastrukturen ble utført den 02.03.21. Klusteret har vært operasjonell i 12 uker i det denne seksjonen skrives. Vår erfaring er at OKD fungerer godt og er pålitelig om systemet ikke har gått ned i en kort periode etter at installasjonen er fullført.

7.0.2 Anbefaling

OKD kan være en god løsning for en konteinerorkestreringsplattform. Den kommer stort sett ferdig konfigurert med mange ulike verktøy. Dette gjør det til et omfattene og stort system med mye funksjonalitet, men det er ikke nødvendigvis like nyttig for alle bruksområder. Det er en bra løsning for å enkelt lage, administrere og lagre konteineriserte programmer på et sted. Det kreves god kompetanse innen Kubernetes for å sette opp sørge for at det er fungerer. Om det ikke er mye kompetanse på kubernetes/OKD i teamet, og det ikke er noen som kan lære seg det kan det være en ide å se videre på Openshift - den betalte versjonen til RedHat som fungerer på samme måte som OKD, men kommer med brukerstøtte.

For arkitektur og skalering av OKD klusteret anbefales det å alltid ha tre kontrollplannoder for administative oppgaver, og minimum en arbeidernode for scheduling av poder for brukere av klusteret. Arbeidernodene kan skaleres opp og ned etter behov for ressurser. Langringsløsningen som er valgt for

systemet, HDFS, er en god løsning for batch jobber og som støtter streaming. For arbeidsoppgaver med streaming finnes det bedre lagringsløsninger som er laget for streaming og er gode på I/O og har lav latency.

7.0.3 Videre arbeid

Videre arbeid består av arbeid gruppen enten har sett på og som kan være relevant å implementere, eller arbeid som ble påbegynt i gjennom prosjektperioden, men ikke ble inkludert i rapporten.

Streaming for Apache Spark med tilhørende lagringsløsning.
Apache spark støtter både batch prosessering og streaming. I denne oppgaver er batch prosessering dokumentert, men systemet skal i utgangspunktet også støtte streaming.

Sikkerhet mellom Apache Spark og HDFS
I den nåværende implementasjonen av prosjektet henter Apache Spark data ut i fra HDFS uten autentisering. Dette oppfyller ikke kravene satt i 2.2.1 og bør utbedres før systemet tas i bruk.

CI/CD pipeline
Det er undersøkt potensielle muligheter for å integrere en CI/CD pipeline inn i OKD klusteret. Et av verktøyene er S2I for å konteinerisere andre applikajoner som kan være ønsket brukt av NTNU SOC.

Test av ytelse
Ytelsestest utført i rapporten har fokusert på test av Dynamisk skalering. Den dokumenterer ikke hvor god ytelse det er på systemtet generelt som kan være nyttig å utføre i fremtiden for å kartlegge ytelsen av plattformen. Gruppen har ikke hatt tid til å sette opp et tilsvarende system for å sammenligne ytelse fra en konteinerisert versjon av Apache Spark med en versjon som ikke bruker konteinere.

Jupyterlab
Jupyterlab er et populært verktøy for utviklere for å lage notatbøker på nett. Det har blitt laget dokumentasjon på hvordan Jupyterlab settes opp, men gruppen har ikke lykkes med å integrere Jupyterlab med Apache spark.

7.0.4 Vurdering av gruppen

I gjennom hele prosjektperioden har gruppen jobben relativt jevnt, med noen naturlige svingninger ettersom andre fag har krevd mer oppmerksomhet. Under prosjektet så har gruppen hatt daglige møter for å oppdatere hverandre på progresjon og mulige utfordringer, og weekly review møter for å diskutere det som har gått bra og det som har gått dårlig. Utover i prosjektperioden

har gruppen ikke fulgt opp weekly reviews like godt. En av grunnene er at vi snakket og jobbet mye sammen hver dag så i slutten av uken da det var tid for weekly review hadde vi ikke så mye å komme med da dette har blitt adressert i løpet av uken.

Gjennomføring av oppgave

Oppgaven inneholder en kombinasjon av mange ulike teknologier og komponenter som skal fungere sammen. Dette har ført til at mye tid er benyttet til å feilsøke og løse ulike problemer.
Under oppsett av test og utviklingsinfrastrukturen der OKD ble satt opp via en automatisk IPI installasjon, satt vi fast på at vi ikke fikk kontakt med HDFS fra internt i klusteret. For å løse problemet ba vi om hjelp fra en systemadministrator av SkyHiGh som hjalp oss med å feilsøke problemet. Det viste seg at under installasjonen så lagde installasjonsprogrammet egne security groups som ikke tillatte kommunikasjon mellom nodene selv om portene var åpne. Dette er et Openstack spesifikt problem som ikke vil være relevant i andre miljø.

Problemene vi erfarte med DNS under testing av dynamisk allokering i kapittel 6.1.3 kan skyldes at vi ikke lærte nok om DNS for å få løst problemet. Vi har hatt flere problemer som dette som har forsinket prosjektet i forhold til planen. Dette har ført til at mye av funksjonalitet og løsninger har blitt tatt med i videre arbeid.

Bibliografi

[1] Apache Software Foundation, *Companies and Organizations using Apache Spark*, Lastet ned 10 Mars 2021. adresse: https://spark.apache.org/powered-by.html.

[2] NTNU, *NTNU SOC - Digital sikkerhet*, Lastet ned 20 Mars 2021. adresse: https://innsida.ntnu.no/wiki/-/wiki/Norsk/NTNU+SOC+-+Digital+sikkerhet.

[3] IBM Cloud Team, *Containers vs. Virtual Machines (VMs): What's the Difference?* Lastet ned 5 April. adresse: https://www.ibm.com/cloud/blog/containers-vs-vms.

[4] VMware, *VMWare History and Interactive Timeline*, Lastet ned 13 Mai 2021. adresse: https://www.vmware.com/timeline.html.

[5] VMware, *Worldwide Virtual Machine Software Market Shares, 2017*, Lastet ned 13 Mai 2021. adresse: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/vmware-idc-virtual-machine-market-shares-2017.pdf.

[6] Wikipedia contributors, *OpenStack — Wikipedia, The Free Encyclopedia*, [Online; Lastet ned 26-February-2021], 2021. adresse: https://en.wikipedia.org/w/index.php?title=OpenStack&oldid=1006548754.

[7] W. contributors, *Big Data Wikipedia*, Lastet ned 13 Mai 2021. adresse: https://en.wikipedia.org/wiki/Big_data.

[8] PWC, *Big Data Hva er Big Data, og hva betyr Big Data for deg?* Lastet ned 13 Mai 2021. adresse: https://www.pwc.no/no/publikasjoner/information-management/big-data.pdf.

[9] Databricks, *The Difference Between Data and Big Data Analytics*, Lastet ned 13 Mai 2021. adresse: https://databricks.com/glossary/big-data-analytics.

[10] Wikipedia, *The Difference Between Data and Big Data Analytics*, Lastet ned 20 April 2021. adresse: https://en.wikipedia.org/w/index.php?title=Apache_Spark&oldid=1016900343.

[11] H. Luu, *Beginning Apache Spark 2: With Resilient Distributed Datasets, Spark SQL, Structured Streaming and Spark Machine Learning Library*, eng, 1. utg. Berkeley, CA: Apress L. P, 2018, isbn: 9781484235782.

[12] Apache, *HDFS Architecture Guide*, Lastet ned 26 April 2021. adresse: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[13] S. Bende og R. Shedge, «Dealing with Small Files Problem in Hadoop Distributed File System,» *Procedia Computer Science*, årg. 79, s. 1001–1012, des. 2016. doi: 10.1016/j.procs.2016.03.127.

[14] *Fedora CoreOS Documentation*, Lastet ned 9 April 2021. adresse: https://docs.fedoraproject.org/en-US/fedora-coreos/.

[15] *Minimum resource requirements*, Lastet ned 24 Februar 2021. adresse: https://docs.okd.io/latest/installing/installing_bare_metal/installing-bare-metal.html#minimum-resource-requirements_installing-bare-metal.

[16] L. Rice, *Container Security: Fundamental Technology Concepts that Protect Containerized Applications 1st Edition*. O'Reilly Media, Inc., 2020, s. 51.

[17] L. Rice, *Container Security: Fundamental Technology Concepts that Protect Containerized Applications 1st Edition*. O'Reilly Media, Inc., 2020, s. 53.

[18] L. Rice, *Container Security: Fundamental Technology Concepts that Protect Containerized Applications 1st Edition*. O'Reilly Media, Inc., 2020, s. 50.

Vedlegg A

Oppgavebeskrivelse

Apache Spark på Openshift Kubernetes Distribution

Prosjektet går ut på å lage et klusterdesign og driftsoppsett som gjør det mulig å kjøre Apache Spark jobber i kontainere på toppen av et OKD kluster (https://www.okd.io) for å få en fleksibel plattform for distribuert avansert sikkerhetsanalyse som eksisterer i kortere perioder. Oppgaven kan inneholde videre testing av ytelse og bør inneholde en anbefaling av skalering av maskinvare og kontainerressurser for å kunne kjøre regnejobber i kontainer.

Oppdragsgiver

Seksjon for Digital sikkerhet ligger under IT-avdelingen og har det helhetlige ansvaret for digital sikkerhet ved NTNU. Seksjonen arbeider proaktivt, aktivt og reaktiv med digital sikkerhet/informasjonssikerhet på flere nivåer i organisasjonen og består av en faggruppe og en rådgivertjeneste. Faggruppen NTNU SOC (Sikkerhetsoperasjonssenter) har ansvaret for deteksjon, sikkerhetsanalyse og hendelseshåndtering mens rådgivertjenesten (Governance, Risk and Compliance - GRC) arbeider med proaktiv sikkerhetsrådgivning, risikostyring og sikkerhetsarkitektur.

Kontaktperson:
Navn: Christoffer Vargtass Hallstensen
Tittel: Gruppeleder SOC
Email: christoffer.hallstensen@ntnu.no
Tek: 611 35 145 / 481 35 180

Oppgavens mål

Målet med oppgaven er designe, installere og konfigurere et produksjonsklart Openshift Kubernetes Distibution (OKD) kluster og bygge kontainere av Apache Spark. Dette for å gi en mer dynamisk og skalerbar kontainer basert plattform for avansert sikkerhetsanalyse på større mengder data i tillegg til nødvendige støttetjenester. Oppgaven kan skaleres opp til å designe en komplett CI/CD pipeline for kontainer baserte applikasjoner og dynamiske analysemiljø dersom gruppen ønsker. Gruppen står også fritt til å komme med forslag til andre ting som ønskes involvert i oppgaven så lenge dette er innenfor opprinnelig avgrenset område.

Oppgavens krav

- Installasjon av et produksjonsklart OKD kluster
- Dokumentere installasjon og drift av et OKD kluster
- Konfigurere Apache Spark til å kjøre i kontainere
- Enkel system og brukerdokumentasjon
- Foreslå arkitektur, skalering og lagring
- Ytelsestest av Apache Spark på OKD
- (Valgfri) Komplett CI

Vedlegg B

Forprosjekt

Prosjektplan for bacheloroppgave 2021

Alf Pettersen,
Anders Moen,
Simon Røe

2021/01/29

Innhold

Figurer

Kapittel 1

Mål og rammer

1.1 Bakgrunn

NTNU Security Operation Center (SOC) er en del av Seksjon for Digital sikkerhet som har ansvaret for den digitale sikkerheten ved NTNU. NTNU SOC sine oppgaver er å detektere hendelser, utføre sikkerhetsanalyse og gjennomføre hendelseshåndtering[1] .
I denne sammenhengen samles det inn store mengder data som inneholder informasjon som kan være nyttig i NTNU SOC sitt sikkerhetsarbeid. Dataene kan analyseres for å utbedre sikkerheten til organisasjonen og videreutvikle rutiner for deteksjon og hendelseshåndtering. For å analysere denne informasjonen ønsker NTNU SOC å benytte seg av en kombinasjon av Openshift Kubernetes Distribution (OKD), som er et program for administrering av konteinere, og Apache Spark, et program for dataanalyse. NTNU SOC ønsker en redegjørelse for om det er en gunstig løsning å kombinere disse to teknologiene for deres brukstilfelle.

1.2 Prosjektmål

1.2.1 Effektmål

Om løsningen viser seg å være hensiktsmessig, kan dette føre til mer effektiv utnyttelse av systemressurser og et sikrere miljø for applikasjoner som kjører i løsningen. I tillegg kan NTNU SOC sin evne til å gjøre operasjoner på store mengder data økes. Rapporten skal kunne brukes som et hjelpemiddel for beslutninger om valg av metode for analyse av større mengder data.

1.2.2 Resultatmål

En vurdering om å kjøre Apache spark i et OKD kluster er en tilfredstillende løsning med tanke på ytelse, drift og skalering. CI/CD skal brukes for automatisk testing og utrulling av kode, og et forslag for baseline-arkitektur med tilhørende ytelsestest og skaleringsmuligheter skal bli presentert. Baseline-arkitekturen skal

inneholde anbefalt forhold mellom CPU, RAM og disk størrelse. Prosjektrapporten skal kunne brukes som en mer omfattende versjon av systemdokumentasjon som går i dybden på metode og beslutninger angående teknologier og prosesser.

1.2.3 Minste brukbare produkt

Minste brukbare produkt er definert som minimumskravet for produktet som skal leveres ved prosjektslutt. Dette innebærer oppsett av et produksjonsklart OKD kluster som kjører Apache Spark i konteinere, samt dokumententasjon av installasjon- og konfigurasjons prosess og enkel brukerdokumentasjon av klusteret. Uten disse komponentene er ikke systemet ansett som brukbart.

1.2.4 Ønsket produkt

I tillegg til punktene nevnt i *1.2.3 Minste brukbare produkt* ønsker vi å implementere CI/CD for integrering og utrulling av konteinerbaserte applikasjoner til OKD. Det er planlagt å utføre skalering med konteinere, men gruppen ønsker å undersøke ytterligere muligheter for skalering av de virtuelle maskinene konteinerene kjører på. Dette for å kunne helautomatisere skalering av infrastrukturen.

1.2.5 Ønsket læringsutbytte

En av hovedgrunnene til at gruppen ønsket å skrive om denne oppgaven, var at vi hadde lyst til å lære mer om ytelsestesting av applikasjoner og automatisering av prosesser og programvare ved bruk av programmerbar infrastruktur. Oppgaven tillater oss å benytte en DevOps-tankegang som vi ønsker å lære mer om, og har som mål om å mestre i løpet av prosjektperioden. Et annet viktig aspekt i oppgaven er sikkerhet. Vi ønsker å undersøke sikkerhet for konteinerne som kjører, og spesielt hvordan OKD har løst sikkerhetsutfordringene knyttet til dette.

1.3 Rammer

1.3.1 Tidsrammer

Frist for innlevering av forprosjekt er satt til den 1. februar og innlevering av prosjektoppgaven skal gjøres innen 20. mai. Oppdragsgiver har ikke spesifisert krav til når det ferdige produktet skal leveres.

1.3.2 Kravspesifikke rammer

I oppgavebeskrivelsen og under møter med veileder og oppdragsgiver har det blitt spesifisert følgende krav til teknologier som skal brukes:

- for administrering av konteinere i infrastrukturen skal OKD benyttes.
- OKD sin installasjonsprosess skal være uavhengig av hvilket skymiljø den skal operere innen.

- Ansible skal brukes for å automatisere installasjonen av OKD.
- for analyse av data, skal en konteinerbasert instans av Apache Spark benyttes.
- Git skal brukes for versjonskontroll.

Kapittel 2

Omfang

2.1 Fagområde

Hovedområdene for oppgaven er infrastruktur som kode og ytelsestesting. Innen infrastruktur som kode vil konfigurasjon av nettverk, lastbalansering, databaser, virtuelle maskiner og konteinere være relevante områder som vi kommer inn på i løpet av oppgaven.

2.2 Oppgavebeskrivelse

Oppgaven går ut på å designe, installere og konfigurere et produksjonsklart OKD kluster for å kjøre konteineriserte applikasjoner. Klusteret skal kjøre konteinere med Apache Spark, som skal brukes til avanserte sikkerhetsanalyser på innsamlede data. Ettersom behovet for ressurser vil variere, er systemet avhengig av å kunne skaleres både opp og ned for å øke ytelse til en oppgave, eller for å gjøre plass til andre oppgaver. Tjenesten vil bli utviklet og testet på NTNU sin skyløsning Skyhigh, men må også omgjøres til å fungere på vSphere for bruk av NTNU SOC. Oppgaven skal komme frem til om det er mulig å bruke en slik tjeneste i produksjon og hvor godt den fungerer.

2.3 Avgrensing

I oppgaven er det spesifisert at OKD og Apache Spark skal benyttes. Derfor vil ikke andre teknologier eller løsninger bli vurdert for administrasjon av konteinere eller dataanalyse. Ettersom løsningen skal fungere uavhengig av hvilken plattform som benyttes, skal ikke skyspesifikke konfigurasjoner være en del av det endelige sluttproduktet. Det skal ikke skrives brukerdokumentasjon eller gjøres rede for bruk av Apache Spark, kun hvordan det settes opp og testes.

Kapittel 3

Prosjektorganisering

3.1 Ansvarsforhold og roller

Bacheloroppgaven er den mest omfattende oppgaven vi har jobbet med hittil i studieløpet. Hvert gruppemedlem er nødt til å ta ansvar for at deloppgaver blir gjennomført på en god måte, samt sørge for kontinuerlig fremdrift i prosjektet. Derfor er det bestemt noen faste roller samtidig som alle er inneforståtte med at hver enkelt må ta ansvar for å oppnå tilstrekkelig fremgang.

3.1.1 Prosjektgruppemedlemmer

- Anders Wormdal Moen (Gruppeleder og scrum master)
- Alf Pettersen (Dokumentansvarlig)
- Simon Rødsbakken Røe (GIT-ansvarlig)

3.1.2 Produkteier

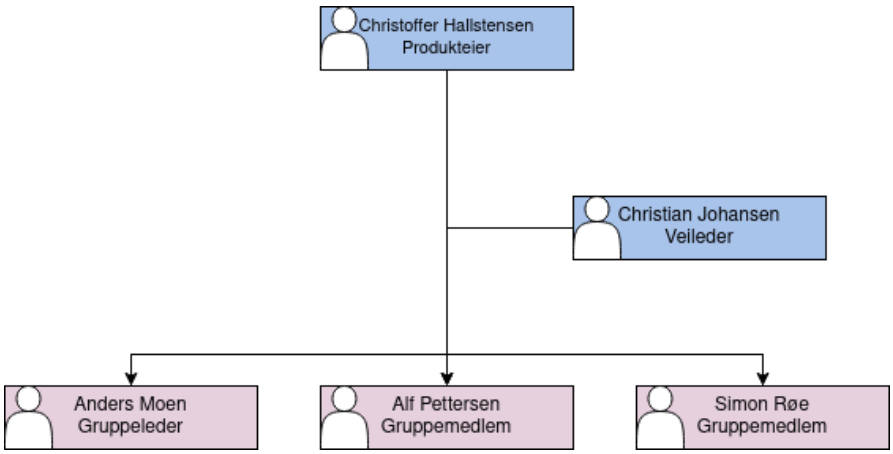
- Christoffer Vargtass Hallstensen arbeider for NTNU SOC og er oppdragsgiver for dette prosjektet.

3.1.3 Veileder

- Christian Johansen - Førsteamanuensis ved NTNU.

3.1.4 Organisasjonskart

Organisasjonskartet er en grafisk representasjon av hvem som er delaktig i dette prosjektet og hvilken rolle vedkommene har.



Figur 3.1: Oversikt over involverte parter i prosjektet.

3.2 Grupperegler

- Det skal jobbes minst 35 timer hver uke inkludert møter og andre prosjektrelaterte oppgaver.
- Dersom man ikke har mulighet til å delta på møte, plikter man til å gå gjennom og lese referat.
- Om en ser tidsfrister man ikke kan opprettholde skal gruppen informeres i god tid om dette.
- Man kan spørre om utsettelse på frister der det skal en forklaring til på hvorfor fristen ikke kan bli holdt. Dette må gjøres i god tid før fristen utløper.
- Møtes til avtalt tid til gruppemøter, veiledermøter og oppdragsgivermøter.
- Om en eller flere gruppemedlemmer ikke har mulighet til å møte opp fysisk, skal resten av gruppen tilrettelegge for å ta i bruk Microsoft Teams.
- Det skal informeres i god tid i forveien eller ha en gyldig grunn ved forsentkomminger over 15 min.
- Rutiner ved gjentatte regelbrudd:
    - Diskusjon i gruppen om problemet og iverksette forbedrende tiltak.
    - Eskalere problemet til veileder og levere en skriftlig advarsel.
    - Tre skriftlige advarsler uten forbedring kvalifiserer til utestenging av gruppen.

3.3 Verktøy

Draw.io	Verktøy for å lage illustrasjoner.
Gitlab	Verktøy for versjonskontroll av kode.
Google Drive	Platform for lagring av dokumenter og notater.
Jira	Benyttes for å administrere og følge opp arbeidsoppgaver.
Microsoft Teams	Møte og kommunikasjonsverktøy mellom studenter, veileder og arbeidsgiver.
Microsoft Visio	Verktøy for å lage illustrasjoner og diagrammer.
Skyhigh	NTNU sin skytjeneste som prosjektet initielt vil bli utviklet og testet på.
Toggl	Verktøy for loggføring av timer brukt på prosjektet.

Kapittel 4

Planlegging, oppfølging og rapportering

4.1 Hovedinndeling av prosjektet

4.1.1 Prosessrammeverk

For å strukturere og planlegge arbeid, har vi valgt å gå for en smidig utviklingsmodell som muliggjør endringer i løpet av prosjektperioden. Vi har valgt å benytte oss av scrum med noen elementer fra kanban. Scrum gir oss ikke fleksibiliteten vi ønsker for å kunne omprioritere oppgaver underveis i en sprint, så en kanban-tavle brukes med pull i stedet for push-baserte oppgaver.

4.1.2 Valg av metode og tilnærming

Alle gruppemedlemmer deltar på identifisering og kartlegging av oppgaver som blir lagt til i en ”to-do” kolonne. Scrum master har ansvar for å sortere oppgavene i en prioritetskø, og kan angi tidsfrist og delegere ansvar ved behov. Videre består kanban-tavlen av kolonnene "in progress"og ”review”, som har begrensninger på antall oppgaver som kan være der samtidig for å redusere antall oppgaver som jobbes med parallelt. I ”review” kolonnen blir ansvaret for oppgaven delegert til et annet gruppemedlem for å kvalitetsjekke tekst eller utføre en gjennomgang av koden. Dette bidrar til å øke kvaliteten på det vi produserer. Etter oppgaven er gjennomgått av et annen gruppemedlem kan den flyttes til den siste kolonnen ”done”. Flere kolonner kan legges til i løpet av prosjektperioden om det sees hensynsfullt.

4.1.3 Plan for statusmøter og beslutningpunkter i perioden

Daily scrum

Det er planlagt 10-15 minutter med oppstartsmøte hver hverdag klokken 08:15 for å oppdatere hverandre på hva man har gjort, hva man skal gjøre og eventuelt

hvilke problemer man står ovenfor. Det tas høyde for at ikke alle har mulighet til å være med hver dag, men da skal dette kommuniseres til gruppen på forhånd. Flere møter for å arbeide i fellesskap vil bli planlagt ut ifra behov.

Veiledermøter

I startfasen av prosjektet er det planlagt å ha ukentlige møter med veileder. For å lykkes med større prosjekter er det essensielt at planleggingen blir gjort grundig for å sette et omfang som er oppnåelig. Derfor anses ukentlig møter med veileder fornuftig for å oppdage feil i planleggingen, og sette fornuftige rammer for resten av prosjektet. Om vi ikke lengre har et behov for hyppige møter kan det reduseres til annenhver uke eller til en ordning som passer oss og veileder.

Møte med oppdragsgiver

Det er ikke planlagt faste møter med oppdragsgiver, men en gruppe er opprettet på Microsoft Teams der meldinger kan bli utvekslet og besvart. Møter med oppdragsgiver planlegges ved behov.

Sprint review / retrospective

Det holdes et kombinert sprint review og retrospective møte basert på arbeid utført i løpet av perioden ved faste tidspunkt hver uke. Ved å avholde møter ofte får gruppen gitt tilbakemeldinger fortløpende som muliggjør forbedring i ett høyere tempo enn om det hadde vært en lengre periode mellom møtene. Møtet er hovedsaklig mellom gruppemedlemmene, men produkteier kan bli invitert ved behov. Produkteier skal bli oppdatert på status og progresjon i gruppen uavhengig av deltagelse på møtet.

Kapittel 5

Organisering av kvalitetssikring

5.1 Dokumentasjon, standarbruk og kildekode

Vi har valgt å gå for en smidig tilnærmingsmetode, og denne modellen dekker ikke kravet for dokumentasjon som oppdragsgiver stiller. Derfor vil dokumentasjon bli utført som selvstendige oppgaver som utføres parallelt ved utvikling av applikasjonen. Kvaliteten på dokumentasjon skal være høy da NTNU SOC har planlagt å gjøre kildekoden offentlig tilgjengelig i fremtiden.

5.2 Konfigurasjonsstyring

For konfigurasjonsstyring vil Git bli benyttet gjennom en Gitlab-instans drevet av NTNU. Det vil bli opprettet repositories for versjonskontroll av både kode skrevet i forbindelse med prosjeket og tekst skrevet i Overleaf.

5.3 Risikoanalyse

Ulike risikoscenario er blitt identifisert og inndelt i prosjekt, teknologi og forretningsbaserte scenario. Videre blir scenarioene kategorisert på grunnlag av antatt sannsynlighet for at hendelsen oppstår, og konsekvens når hendelsen inntreffer. Tiltakspakke er utarbeidet for scenario med en samlet score over fire for å mitigere risiko. En totalsum for risiko for hvert scenario blir kalkulert ved bruk av formelen $Risiko = Sannsynlighet\ x\ Konsekvens$. Skala for sannsynlighet går fra *Lite sannsynlig*, som gir ett poeng opp til *Meget sannsynlig*, som gir fire poeng. Samme skala gjelder for beregning av risiko der *Liten* konsekvens gir ett poeng og *Kritisk* gir fire poeng.

5.3.1 Risikoscenarioer

Prosjekt

Scenario 1:
Gruppemedlem legger ikke inn påkrevd arbeidsmengde eller leverer ikke forventet kvalitet på arbeid grunnet lav motivasjon. Fører til at gruppen blir for sen i forhold til arbeidsoppgaver.

Scenario 2:
Gruppemedlem blir syk eller utilgjengelig for et kort periode. Fører til mild eller moderat forsinkelse i prosjektplan.

Scenario 3:
Gruppemedlem blir syk eller utilgjengelig for en lengre periode. Fører til moderat til høy forsinkelse i prosjektplan.

Scenario 4:
Gruppemedlemmenes manglende kompetanse innen planlegging og estimering resulterer i en prosjektplan som enten ikke er gjennomførbar eller legger opp til for lite arbeid. Fører til et mangelfult og dårlig produkt.

Scenario 5:
Gruppemedlemmenes forskjellige tankemåte fører til uenighet om hvilken fremgangsmåte som skal bli brukt for å løse et gitt problem. Dette kan føre til en intern strid i gruppen og senke moral betraktelig.

Scenario 6
Veileder eller oppdragsgiver blir syk eller utilgjengelig for spørsmål eller møter i en lengre periode. Dette medfører mindre støtte og at signifikante beslutninger kan måtte tas innad i gruppen.

Teknologi

Scenario 7:
Gruppemedlemmer mister tilgang til loggfiler, planleggingsdokumenter eller dokumentasjon, siden en eller flere tjenester benyttet blir utilgjengelig. Fører til mindre effektivt arbeid og i verste fall tap av data.

Scenario 8:
Openstack og/eller vSphere blir utilgjengelig. Fører til store deler av oppgaven ikke er gjennomførbar.

Forretning

Scenario 9:
Gruppemedlems uaktsomhet fører til lekkasje av konfidensiell data gitt fra oppdragsgiver. Dette kan medføre omdømmetap og en potensiell trusselaktør kan tilegne seg informasjon som kan bli benyttet i et angrep på et senere tidspunkt.

5.3.2 Risikomatrise før tiltaksplan

Sannsynlighet → Konsekvens ↓	**Lite sannsynlig**	**Mindre sannsynlig**	**Sannsynlig**	**Meget sannsynlig**
Liten			1	
Moderat		2, 6		5
Alvorlig			7	
Kritisk	3	8, 9	4	

Tabell 1: Risikomatrise før tiltaksplan.

5.3.3 Tiltaksplan

Scenario	Tiltak
Nr.3	Samtale med veileder om problemet og muligheter videre. Med langvarig nedjustert kapasitet kan fremdriftsplanen måtte endres for å bli ferdig med oppgaven.
Nr.4	Sette av mye tid på planlegging og sette seg inn i ulike fasene av oppgaven for å få en bedre oversikt og gi mer grunnlag for å levere et godt estimat på tidsbruk. På momenter som gruppen har lite grunnlagt for å angi et godt estimat kan ekstern hjelp fra en ekspert være en mulighet.
Nr.5	I en diskusjon vil flertallet bestemme fremgangsmåte ved håndsopprekning. Det vil også bli planlagt og arrangert ulike teambuilding og sosiale aktiviteter for å bygge moral og samhold i gruppen.
Nr.6	Ta ukentlige sikkerhetskopier av arbeidsdokumenter og filer, timelogger og referat. Prosjektrapporten skal tas sikkerhetskopi av hver dag.
Nr.7	Diskutere med oppdragsgiver om mulighet for alternative skyløsninger, f.eks Amazon Web Services eller Microsoft Azure.
Nr.8	Utarbeide retningslinjer ved håndtering av konfidensiell informasjon med tanke på både bruk og lagring av informasjonen.

5.3.4 Risikomatrise etter tiltaksplan

Sannsynlighet → Konsekvens ↓	**Lite sannsynlig**	**Mindre sannsynlig**	**Sannsynlig**	**Meget sannsynlig**
Liten		7	1, 5	
Moderat		2, 6, 8, 9		
Alvorlig	3	4		
Kritisk				

Tabell 2: Risikomatrise etter tiltaksplan.

5.3.5 Akseptabel risiko

Scenario med fire eller mindre i totalsum er definert som akseptabel risiko. Etter tiltaksplan har alle scenarioene en totalsum på fire eller mindre, eksklusivt scenario fire. Etter nøye vurderinger er også scenario fire ansett som akseptabel, og en risiko gruppen er villig til å ta uten videre tiltak.

Kapittel 6

Plan for gjennomføring

6.1 Inndeling av prosjekt

Prosjektet er delt inn i fire hovedfaser der fase nummer tre er avhengig av at fase to er gjennomført. Den siste fasen inneholder ingen nye målsetninger eller oppgaver, men er til for å kunne ferdigstille det arbeidet som allerede er påbegynt.

6.1.1 Utvikling og implementering

Målet etter at denne fasen av prosjektet er ferdig, er å få et funksjonelt og fungerende OKD kluster med Apache Spark konteinere. Første steg for å oppnå dette er å utvikle en konseptutprøvning ved bruk av Skyhigh, som deretter bygges videre på i gjennom automatiseringsprosesser og med støtte for CI/CD. Løsningen skal også være kompatibel med vSphere der det ferdige produktet til slutt skal kjøre i produksjon. Kontinuerlig gjennom denne fasen skal arbeidet dokumenteres som gir grunnlag for å levere en god og fullstendig dokumentasjon.

6.1.2 Testing og utredning

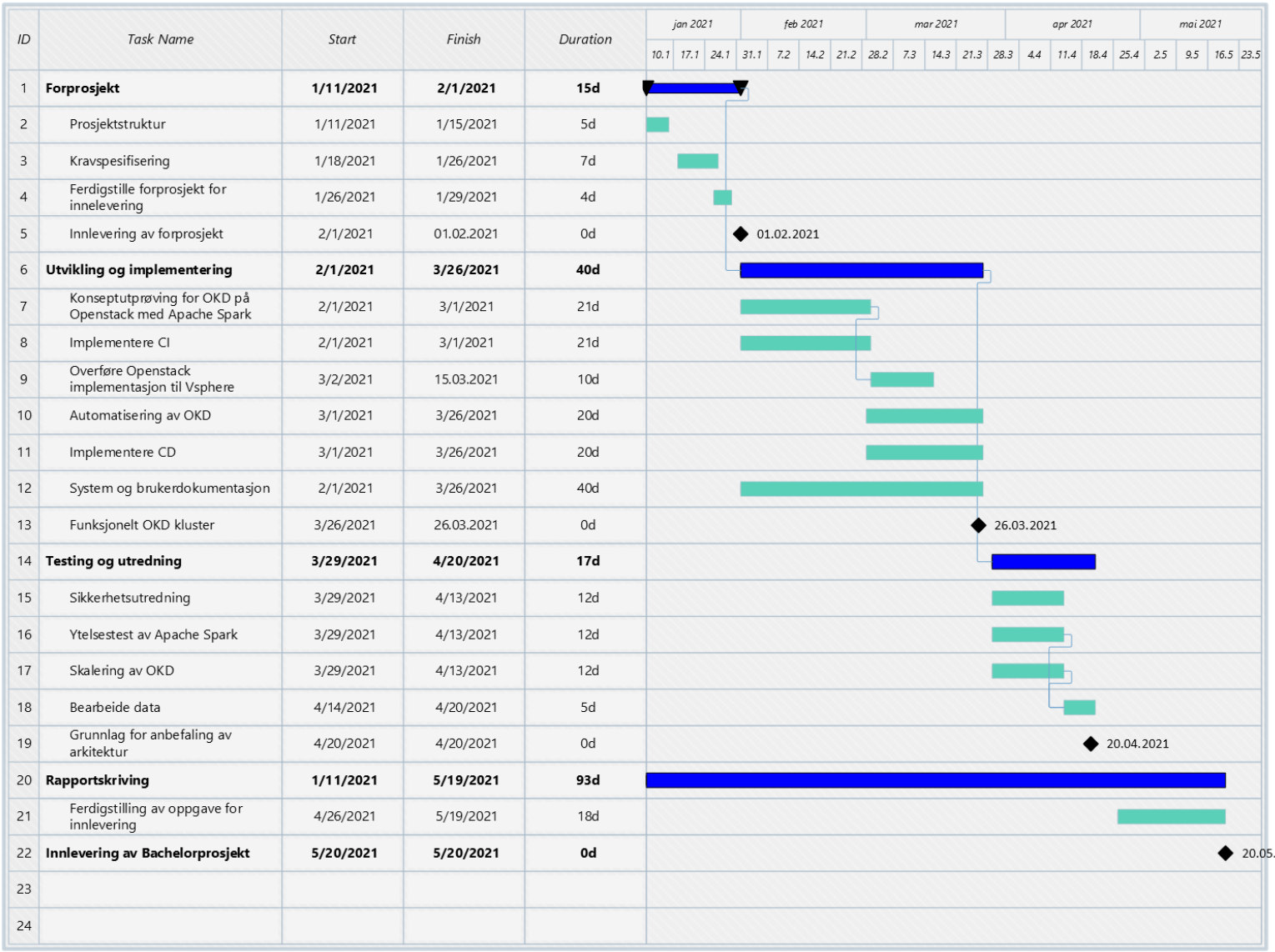
For å kunne anbefale en arkitektur skal det gjennomføres tester for robusthet, skalering av infrastruktur og ytelse på applikasjonsnivå. Det er planlagt tid til analyse og bearbeiding av data generert fra testene. Disse dataene har høy innvirkning på hvilken arkitektur som viser seg å være mest hensiktsmessig.

6.1.3 Rapportskriving

Under prosjektperioden kommer gruppen til å jobbe kontinuerlig med prosjektrapporten, samtidig som vi følger planen beskrevet i Gantt-diagrammet. Siden det er planlagt mye arbeid over relativt kort tid, er ikke nødvendigvis prosjektrapporten ferdigskrevet til den 20. april, som er planlagt dato for at den praktiske delen av oppgaven skal være ferdig. Den siste fasen er dedikert til å finskrive rapporten, og å utføre mindre oppgaver vi ikke hadde tid til for å holde følge med planlagt progresjon.

6.2 Gantt-diagram

Diagrammet viser et estimat for start- og sluttdato for når de forskjellige delaktivitetene i prosjektet skal utføres. Dette vil fungere som en pekepinn på hvor langt gruppen har kommet i prosjektfasen.



ID	Task Name	Start	Finish	Duration
1	**Forprosjekt**	**1/11/2021**	**2/1/2021**	**15d**
2	Prosjektstruktur	1/11/2021	1/15/2021	5d
3	Kravspesifisering	1/18/2021	1/26/2021	7d
4	Ferdigstille forprosjekt for innelevering	1/26/2021	1/29/2021	4d
5	Innlevering av forprosjekt	2/1/2021	01.02.2021	0d
6	**Utvikling og implementering**	**2/1/2021**	**3/26/2021**	**40d**
7	Konseptutprøving for OKD på Openstack med Apache Spark	2/1/2021	3/1/2021	21d
8	Implementere CI	2/1/2021	3/1/2021	21d
9	Overføre Openstack implementasjon til Vsphere	3/2/2021	15.03.2021	10d
10	Automatisering av OKD	3/1/2021	3/26/2021	20d
11	Implementere CD	3/1/2021	3/26/2021	20d
12	System og brukerdokumentasjon	2/1/2021	3/26/2021	40d
13	Funksjonelt OKD kluster	3/26/2021	26.03.2021	0d
14	**Testing og utredning**	**3/29/2021**	**4/20/2021**	**17d**
15	Sikkerhetsutredning	3/29/2021	4/13/2021	12d
16	Ytelsestest av Apache Spark	3/29/2021	4/13/2021	12d
17	Skalering av OKD	3/29/2021	4/13/2021	12d
18	Bearbeide data	4/14/2021	4/20/2021	5d
19	Grunnlag for anbefaling av arkitektur	4/20/2021	4/20/2021	0d
20	**Rapportskriving**	**1/11/2021**	**5/19/2021**	**93d**
21	Ferdigstilling av oppgave for innlevering	4/26/2021	5/19/2021	18d
22	**Innlevering av Bachelorprosjekt**	**5/20/2021**	**5/20/2021**	**0d**
23				
24				

Figur 6.1: Gantt-diagram

Bibliografi

[1] NTNU. adresse: https://innsida.ntnu.no/wiki/-/wiki/Norsk/NTNU+SOC+-+Digital+sikkerhet.

Vedlegg C

Apache Spark

C.1 docker-image-tool.sh

```
#!/usr/bin/env bash

# Licensed to the Apache Software Foundation (ASF) under one or
    more
# contributor license agreements.  See the NOTICE file
    distributed with
# this work for additional information regarding copyright
    ownership.
# The ASF licenses this file to You under the Apache License,
    Version 2.0
# (the "License"); you may not use this file except in compliance
    with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
    software
# distributed under the License is distributed on an "AS IS"
    BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied.
# See the License for the specific language governing permissions
    and
# limitations under the License.
#

# This script builds and pushes docker images when run from a
    release of Spark
# with Kubernetes support.

function error {
  echo "$@" 1>&2
  exit 1
}

if [ -z "${SPARK_HOME}" ]; then
  SPARK_HOME="$(cd␣"‘dirname "$0"‘"/..;␣pwd)"
```

```
fi
. "${SPARK_HOME}/bin/load-spark-env.sh"

CTX_DIR="$SPARK_HOME/target/tmp/docker"

function is_dev_build {
  [ ! -f "$SPARK_HOME/RELEASE" ]
}

function cleanup_ctx_dir {
  if is_dev_build; then
    rm -rf "$CTX_DIR"
  fi
}

trap cleanup_ctx_dir EXIT

function image_ref {
  local image="$1"
  local add_repo="${2:-1}"
  if [ $add_repo = 1 ] && [ -n "$REPO" ]; then
    image="$REPO/$image"
  fi
  if [ -n "$TAG" ]; then
    image="$image:$TAG"
  fi
  echo "$image"
}

function docker_push {
  local image_name="$1"
  if [ ! -z $(docker images -q "$(image_ref␣${image_name})") ];
      then
    docker push "$(image_ref␣${image_name})"
    if [ $? -ne 0 ]; then
      error "Failed␣to␣push␣$image_name␣Docker␣image."
    fi
  else
    echo "$(image_ref␣${image_name})␣image␣not␣found.␣Skipping␣
        push␣for␣this␣image."
  fi
}

function resolve_file {
  local FILE=$1
  if [ -n "$FILE" ]; then
    local DIR=$(dirname $FILE)
    DIR=$(cd $DIR && pwd)
    FILE="${DIR}/$(basename␣$FILE)"
  fi
  echo $FILE
}

# Create a smaller build context for docker in dev builds to make
    the build faster. Docker
# uploads all of the current directory to the daemon, and it can
   get pretty big with dev
# builds that contain test log files and other artifacts.
#
```

```
# Three build contexts are created, one for each image: base,
    pyspark, and sparkr. For them
# to have the desired effect, the docker command needs to be
    executed inside the appropriate
# context directory.
#
# Note: docker does not support symlinks in the build context.
function create_dev_build_context {(
  set -e
  local BASE_CTX="$CTX_DIR/base"
  mkdir -p "$BASE_CTX/kubernetes"
  cp -r "resource-managers/kubernetes/docker/src/main/dockerfiles
      " \
    "$BASE_CTX/kubernetes/dockerfiles"

  cp -r "assembly/target/scala-$SPARK_SCALA_VERSION/jars" "
      $BASE_CTX/jars"
  cp -r "resource-managers/kubernetes/integration-tests/tests" \
    "$BASE_CTX/kubernetes/tests"

  mkdir "$BASE_CTX/examples"
  cp -r "examples/src" "$BASE_CTX/examples/src"
  # Copy just needed examples jars instead of everything.
  mkdir "$BASE_CTX/examples/jars"
  for i in examples/target/scala-$SPARK_SCALA_VERSION/jars/*; do
    if [ ! -f "$BASE_CTX/jars/$(basename␣$i)" ]; then
      cp $i "$BASE_CTX/examples/jars"
    fi
  done

  for other in bin sbin data; do
    cp -r "$other" "$BASE_CTX/$other"
  done

  local PYSPARK_CTX="$CTX_DIR/pyspark"
  mkdir -p "$PYSPARK_CTX/kubernetes"
  cp -r "resource-managers/kubernetes/docker/src/main/dockerfiles
      " \
    "$PYSPARK_CTX/kubernetes/dockerfiles"
  mkdir "$PYSPARK_CTX/python"
  cp -r "python/lib" "$PYSPARK_CTX/python/lib"
  cp -r "python/pyspark" "$PYSPARK_CTX/python/pyspark"

  local R_CTX="$CTX_DIR/sparkr"
  mkdir -p "$R_CTX/kubernetes"
  cp -r "resource-managers/kubernetes/docker/src/main/dockerfiles
      " \
    "$R_CTX/kubernetes/dockerfiles"
  cp -r "R" "$R_CTX/R"
)}

function img_ctx_dir {
  if is_dev_build; then
    echo "$CTX_DIR/$1"
  else
    echo "$SPARK_HOME"
  fi
}

function build {
```

```
  local BUILD_ARGS
  local SPARK_ROOT="$SPARK_HOME"

  if is_dev_build; then
    create_dev_build_context || error "Failed␣to␣create␣docker␣
        build␣context."
    SPARK_ROOT="$CTX_DIR/base"
  fi

  # Verify that the Docker image content directory is present
  if [ ! -d "$SPARK_ROOT/kubernetes/dockerfiles" ]; then
    error "Cannot␣find␣docker␣image.␣This␣script␣must␣be␣run␣from
        ␣a␣runnable␣distribution␣of␣Apache␣Spark."
  fi

  # Verify that Spark has actually been built/is a runnable
      distribution
  # i.e. the Spark JARs that the Docker files will place into the
      image are present
  local TOTAL_JARS=$(ls $SPARK_ROOT/jars/spark-* | wc -l)
  TOTAL_JARS=$(( $TOTAL_JARS ))
  if [ "${TOTAL_JARS}" -eq 0 ]; then
    error "Cannot␣find␣Spark␣JARs.␣This␣script␣assumes␣that␣
        Apache␣Spark␣has␣first␣been␣built␣locally␣or␣this␣is␣a␣
        runnable␣distribution."
  fi

  local BUILD_ARGS=(${BUILD_PARAMS})

  # If a custom SPARK_UID was set add it to build arguments
  if [ -n "$SPARK_UID" ]; then
    BUILD_ARGS+=(--build-arg spark_uid=$SPARK_UID)
  fi

  local BINDING_BUILD_ARGS=(
    ${BUILD_ARGS[@]}
    --build-arg
    base_img=$(image_ref spark)
  )

  local BASEDOCKERFILE=${BASEDOCKERFILE:-"kubernetes/dockerfiles/
      spark/Dockerfile"}
  local PYDOCKERFILE=${PYDOCKERFILE:-false}
  local RDOCKERFILE=${RDOCKERFILE:-false}
  local ARCHS=${ARCHS:-"--platform␣linux/amd64,linux/arm64"}

  (cd $(img_ctx_dir base) && docker build $NOCACHEARG "${
      BUILD_ARGS[@]}" \
    -t $(image_ref spark) \
    -f "$BASEDOCKERFILE" .)
  if [ $? -ne 0 ]; then
    error "Failed␣to␣build␣Spark␣JVM␣Docker␣image,␣please␣refer␣
        to␣Docker␣build␣output␣for␣details."
  fi
  if [ "${CROSS_BUILD}" != "false" ]; then
  (cd $(img_ctx_dir base) && docker buildx build $ARCHS
      $NOCACHEARG "${BUILD_ARGS[@]}" --push \
    -t $(image_ref spark) \
    -f "$BASEDOCKERFILE" .)
  fi
```

```
  if [ "${PYDOCKERFILE}" != "false" ]; then
    (cd $(img_ctx_dir pyspark) && docker build $NOCACHEARG "${
        BINDING_BUILD_ARGS[@]}" \
      -t $(image_ref spark-py) \
      -f "$PYDOCKERFILE" .)
      if [ $? -ne 0 ]; then
        error "Failed␣to␣build␣PySpark␣Docker␣image,␣please␣refer
            ␣to␣Docker␣build␣output␣for␣details."
      fi
      if [ "${CROSS_BUILD}" != "false" ]; then
        (cd $(img_ctx_dir pyspark) && docker buildx build $ARCHS
            $NOCACHEARG "${BINDING_BUILD_ARGS[@]}" --push \
          -t $(image_ref spark-py) \
          -f "$PYDOCKERFILE" .)
      fi
  fi

  if [ "${RDOCKERFILE}" != "false" ]; then
    (cd $(img_ctx_dir sparkr) && docker build $NOCACHEARG "${
        BINDING_BUILD_ARGS[@]}" \
      -t $(image_ref spark-r) \
      -f "$RDOCKERFILE" .)
    if [ $? -ne 0 ]; then
      error "Failed␣to␣build␣SparkR␣Docker␣image,␣please␣refer␣to
          ␣Docker␣build␣output␣for␣details."
    fi
    if [ "${CROSS_BUILD}" != "false" ]; then
      (cd $(img_ctx_dir sparkr) && docker buildx build $ARCHS
          $NOCACHEARG "${BINDING_BUILD_ARGS[@]}" --push \
        -t $(image_ref spark-r) \
        -f "$RDOCKERFILE" .)
    fi
  fi
}

function push {
  docker_push "spark"
  docker_push "spark-py"
  docker_push "spark-r"
}

function usage {
  cat <<EOF
Usage: $0 [options] [command]
Builds or pushes the built-in Spark Docker image.

Commands:
  build       Build image. Requires a repository address to be
      provided if the image will be
              pushed to a different registry.
  push        Push a pre-built image to a registry. Requires a
      repository address to be provided.

Options:
  -f file               Dockerfile to build for JVM based Jobs.
      By default builds the Dockerfile shipped with Spark.
  -p file               (Optional) Dockerfile to build for
      PySpark Jobs. Builds Python dependencies and ships with
      Spark.
```

```
                        Skips building PySpark docker image if
                            not specified.
  -R file               (Optional) Dockerfile to build for SparkR
      Jobs. Builds R dependencies and ships with Spark.
                        Skips building SparkR docker image if not
                             specified.
  -r repo               Repository address.
  -t tag                Tag to apply to the built image, or to
     identify the image to be pushed.
  -m                    Use minikube's␣Docker␣daemon.
␣␣-n␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Build␣docker␣image␣with␣--no-cache
␣␣-u␣uid␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣UID␣to␣use␣in␣the␣USER␣directive␣to␣set␣
    the␣user␣the␣main␣Spark␣process␣runs␣as␣inside␣the
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣resulting␣container
␣␣-X␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Use␣docker␣buildx␣to␣cross␣build.␣
    Automatically␣pushes.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣See␣https://docs.docker.com/buildx/
    working-with-buildx/␣for␣steps␣to␣setup␣buildx.
␣␣-b␣arg␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Build␣arg␣to␣build␣or␣push␣the␣image.␣For
    ␣multiple␣build␣args,␣this␣option␣needs␣to
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣be␣used␣separately␣for␣each␣build␣arg.

Using␣minikube␣when␣building␣images␣will␣do␣so␣directly␣into␣
    minikube's Docker daemon.
There is no need to push the images into minikube in that case,
    they'll␣be␣automatically
available␣when␣running␣applications␣inside␣the␣minikube␣cluster.

Check␣the␣following␣documentation␣for␣more␣information␣on␣using␣
    the␣minikube␣Docker␣daemon:

␣␣https://kubernetes.io/docs/getting-started-guides/minikube/#
    reusing-the-docker-daemon

Examples:
␣␣-␣Build␣image␣in␣minikube␣with␣tag␣"testing"
␣␣␣␣$0␣-m␣-t␣testing␣build

␣␣-␣Build␣PySpark␣docker␣image
␣␣␣␣$0␣-r␣docker.io/myrepo␣-t␣v2.3.0␣-p␣kubernetes/dockerfiles/
    spark/bindings/python/Dockerfile␣build

␣␣-␣Build␣and␣push␣image␣with␣tag␣"v2.3.0"␣to␣docker.io/myrepo
␣␣␣␣$0␣-r␣docker.io/myrepo␣-t␣v2.3.0␣build
␣␣␣␣$0␣-r␣docker.io/myrepo␣-t␣v2.3.0␣push

␣␣-␣Build␣and␣push␣JDK11-based␣image␣with␣tag␣"v3.0.0"␣to␣docker.
    io/myrepo
␣␣␣␣$0␣-r␣docker.io/myrepo␣-t␣v3.0.0␣-b␣java_image_tag=11-jre-
    slim␣build
␣␣␣␣$0␣-r␣docker.io/myrepo␣-t␣v3.0.0␣push

␣␣-␣Build␣and␣push␣JDK11-based␣image␣for␣multiple␣archs␣to␣docker
    .io/myrepo
␣␣␣␣$0␣-r␣docker.io/myrepo␣-t␣v3.0.0␣-X␣-b␣java_image_tag=11-jre-
    slim␣build
␣␣␣␣#␣Note:␣buildx,␣which␣does␣cross␣building,␣needs␣to␣do␣the␣
    push␣during␣build
␣␣␣␣#␣So␣there␣is␣no␣separate␣push␣step␣with␣-X
```

```
EOF
}

if␣[[␣"$@"␣=␣*--help␣]]␣||␣[[␣"$@"␣=␣*-h␣]];␣then
␣␣usage
␣␣exit␣0
fi

REPO=
TAG=
BASEDOCKERFILE=
PYDOCKERFILE=
RDOCKERFILE=
NOCACHEARG=
BUILD_PARAMS=
SPARK_UID=
CROSS_BUILD="false"
while␣getopts␣f:p:R:mr:t:Xnb:u:␣option
do
␣case␣"${option}"
␣in
␣f)␣BASEDOCKERFILE=$(resolve_file␣${OPTARG});;
␣p)␣PYDOCKERFILE=$(resolve_file␣${OPTARG});;
␣R)␣RDOCKERFILE=$(resolve_file␣${OPTARG});;
␣r)␣REPO=${OPTARG};;
␣t)␣TAG=${OPTARG};;
␣n)␣NOCACHEARG="--no-cache";;
␣b)␣BUILD_PARAMS=${BUILD_PARAMS}"␣--build-arg␣"${OPTARG};;
␣X)␣CROSS_BUILD=1;;
␣m)
␣␣␣if␣!␣which␣minikube␣1>/dev/null;␣then
␣␣␣␣␣error␣"Cannot␣find␣minikube."
␣␣␣fi
␣␣␣if␣!␣minikube␣status␣1>/dev/null;␣then
␣␣␣␣␣error␣"Cannot␣contact␣minikube.␣Make␣sure␣it's running."
␣␣␣fi
␣␣␣eval␣$(minikube␣docker-env␣--shell␣bash)
␣␣␣;;
␣␣u)␣SPARK_UID=${OPTARG};;
␣esac
done

case␣"${@: -1}"␣in
␣␣build)
␣␣␣␣build
␣␣␣␣;;
␣␣push)
␣␣␣␣if␣[␣-z␣"$REPO"␣];␣then
␣␣␣␣␣␣usage
␣␣␣␣␣␣exit␣1
␣␣␣␣fi
␣␣␣␣push
␣␣␣␣;;
␣␣*)
␣␣␣␣usage
␣␣␣␣exit␣1
␣␣␣␣;;
esac
```

C.2 Dockerfile

```
#
# Licensed to the Apache Software Foundation (ASF) under one or
    more
# contributor license agreements.  See the NOTICE file
    distributed with
# this work for additional information regarding copyright
    ownership.
# The ASF licenses this file to You under the Apache License,
    Version 2.0
# (the "License"); you may not use this file except in compliance
     with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
    software
# distributed under the License is distributed on an "AS IS"
    BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
     implied.
# See the License for the specific language governing permissions
     and
# limitations under the License.
#
ARG java_image_tag=11-jre-slim

FROM openjdk:${java_image_tag}

ARG spark_uid=185

# Before building the docker image, first build and make a Spark
    distribution following
# the instructions in http://spark.apache.org/docs/latest/
    building-spark.html.
# If this docker file is being used in the context of building
    your images from a Spark
# distribution, the docker build command should be invoked from
    the top level directory
# of the Spark distribution. E.g.:
# docker build -t spark:latest -f kubernetes/dockerfiles/spark/
    Dockerfile .

RUN set -ex && \
    sed -i 's/http:\/\/deb.\(.*\)/https:\/\/deb.\1/g' /etc/apt/
        sources.list && \
    apt-get update && \
    ln -s /lib /lib64 && \
    apt install -y bash tini libc6 libpam-modules krb5-user
        libnss3 procps && \
    mkdir -p /opt/spark && \
    mkdir -p /opt/spark/examples && \
    mkdir -p /opt/spark/work-dir && \
    touch /opt/spark/RELEASE && \
    rm /bin/sh && \
    ln -sv /bin/bash /bin/sh && \
```

```
    echo "auth␣required␣pam_wheel.so␣use_uid" >> /etc/pam.d/su &&
        \
    chgrp root /etc/passwd && chmod ug+rw /etc/passwd && \
    rm -rf /var/cache/apt/*

COPY jars /opt/spark/jars
COPY bin /opt/spark/bin
COPY sbin /opt/spark/sbin
COPY kubernetes/dockerfiles/spark/entrypoint.sh /opt/
COPY kubernetes/dockerfiles/spark/decom.sh /opt/
COPY examples /opt/spark/examples
COPY kubernetes/tests /opt/spark/tests
COPY data /opt/spark/data

ENV SPARK_HOME /opt/spark

WORKDIR /opt/spark/work-dir
RUN chmod g+w /opt/spark/work-dir
RUN chmod a+x /opt/decom.sh

ENTRYPOINT [ "/opt/entrypoint.sh" ]

# Specify the User that the actual main process will run as
USER ${spark_uid}
```

C.3 Dockerfile med Python støtte

```
#
# Licensed to the Apache Software Foundation (ASF) under one or
    more
# contributor license agreements.  See the NOTICE file
    distributed with
# this work for additional information regarding copyright
    ownership.
# The ASF licenses this file to You under the Apache License,
    Version 2.0
# (the "License"); you may not use this file except in compliance
     with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
    software
# distributed under the License is distributed on an "AS IS"
    BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
     implied.
# See the License for the specific language governing permissions
     and
# limitations under the License.
#

ARG base_img

FROM $base_img
WORKDIR /
```

```
# Reset to root to run installation tasks
USER 0

RUN mkdir ${SPARK_HOME}/python
RUN apt-get update && \
    apt install -y python3 python3-pip hdfs && \
    pip3 install --upgrade pip setuptools && \
    # Removed the .cache to save space
    rm -r /root/.cache && rm -rf /var/cache/apt/*

COPY python/pyspark ${SPARK_HOME}/python/pyspark
COPY python/lib ${SPARK_HOME}/python/lib

WORKDIR /opt/spark/work-dir
ENTRYPOINT [ "/opt/entrypoint.sh" ]

# Specify the User that the actual main process will run as
ARG spark_uid=185
USER ${spark_uid}
```

C.4 Spark operator applikasjon

```
#
# Copyright 2018 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License
    .
# You may obtain a copy of the License at
#
#     https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
    software
# distributed under the License is distributed on an "AS IS"
    BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied.
# See the License for the specific language governing permissions
    and
# limitations under the License.
#
# Support for Python is experimental, and requires building
    SNAPSHOT image of Apache Spark,
# with `imagePullPolicy` set to Always

apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: test1
  namespace: default
spec:
  sparkConf:
    "spark.ui.port": "4045"
    "spark.eventLog.enabled": "true"
    "spark.eventLog.dir": "hdfs://dnshost.internt:9000/user/
       fedora/sparklogs"
  hadoopConf:
    "dfs.datanode.http.address": dnshost.internt
    "dfs.datanode.address": dnshost.internt:9866
    "dfs.datanode.http.address": dnshost.internt:9864
    "dfs.datanode.ipc.address": dnshost.internt:9867
    "dfs.client.use.datanode.hostname": "true"
    "fs.defaultFS": hdfs://dnshost.internt:9000
    "hadoop.tmp.dir": /home/fedora/hdfs
  type: Python
  pythonVersion: "2"
  mode: cluster
  image: "docker.io/anderwm/spark:pythonv4"
  imagePullPolicy: Always
  mainApplicationFile: http://10.212.139.154:4444/testwc.py
  sparkVersion: "3.0.0"
  dynamicAllocation:
    enabled: true
    initialExecutors: 1
    minExecutors: 1
    maxExecutors: 5
  driver:
    dnsConfig:
```

```
      nameservers:
        - 10.0.3.155
      options:
        - name: ndots
          value: "1"
        - name: edns0
    cores: 1
    coreLimit: "1200m"
    memory: "4000m"
    labels:
      version: 3.0.0
    serviceAccount: spark
  executor:
    cores: 1
    instances: 1
    memory: "2500m"
    dnsConfig:
      nameservers:
        - 10.0.3.155
      options:
        - name: ndots
          value: "1"
        - name: edns0
    labels:
      version: 3.0.0
```

Vedlegg D

OKD konfigurasjonsfiler

D.1 Konfigurasjonsfiler for støttetjenester

D.1.1 HAProxy konfigurasjon

Kodeliste D.1: Listing

```
# Global settings
#
    ---------------------------------------------------------------------

global
    maxconn     20000
    log         /dev/log local0 info
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    user        haproxy
    group       haproxy
    daemon

    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats

#
    ---------------------------------------------------------------------

# common defaults that all the 'listen' and 'backend' sections
    will
# use if not designated in their block
#
    ---------------------------------------------------------------------

defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
```

```
    timeout queue           1m
    timeout connect         10s
    timeout client          300s
    timeout server          300s
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 20000

listen stats
    bind :9000
    mode http
    stats enable
    stats uri /

# Kubernetes API
frontend kubernetes_api_frontend
    bind :6443
    default_backend kubernetes_api_backend
    mode tcp
    option tcplog

backend kubernetes_api_backend
    balance source
    mode tcp
# Bootstrap can be removed after the cluster is initialized
    server      bootstrap 10.0.10.11:6443 check
    server      controll-plane-1 10.0.10.12:6443 check
    server      controll-plane-2 10.0.10.13:6443 check
    server      controll-plane-3 10.0.10.14:6443 check


# Machine config server
frontend machine_config_server_frontend
    bind :22623
    default_backend machine_config_server_backend
    mode tcp
    option tcplog

backend machine_config_server_backend
    balance source
    mode tcp
# Bootstrap can be removed after the cluster is initialized
    server      bootstrap 10.0.10.11:22623 check
    server      controll-plane-1 10.0.10.12:22623 check
    server      controll-plane-2 10.0.10.13:22623 check
    server      controll-plane-3 10.0.10.14:22623 check


# HTTP
frontend http_ingress_frontend
    bind :80
    default_backend http_ingress_backend
    mode tcp
    option tcplog

backend http_ingress_backend
    balance source
    mode tcp
# If controll plane nodes are scheduleable the needs to be added
    here.
```

```
    server      compute-1 10.0.10.15:80 check
    server      compute-2 10.0.10.16:80 check


# HTTPS
frontend https_ingress_frontend
    bind *:443
    default_backend http_ingress_backend
    mode tcp
    option tcplog

backend http_ingress_backend
    balance source
    mode tcp
# If controll plane nodes are scheduleable the needs to be added
   here.
    server      compute-1 10.0.10.15:443 check
    server      compute-2 10.0.10.16:443 check
```

D.1.2 DNS konfigurasjon

named.conf

```
   //
// named.conf
//
// Provided by Red Hat bind package to configure the ISC BIND named(8) DNS
// server as a caching only nameserver (as a localhost DNS resolver only).
//
// See /usr/share/doc/bind*/sample/ for example named configuration files.
//
// See the BIND Administrator's Reference Manual (ARM) for details about the
// configuration located in /usr/share/doc/bind-{version}/Bv9ARM.html

options {
listen-on port 53 { 127.0.0.1; 10.0.10.10; }; // Added the IP of the service node
# listen-on-v6 port 53 { ::1; };
directory  "/var/named";
dump-file  "/var/named/data/cache_dump.db";
statistics-file "/var/named/data/named_stats.txt";
memstatistics-file "/var/named/data/named_mem_stats.txt";
recursing-file  "/var/named/data/named.recursing";
secroots-file   "/var/named/data/named.secroots";
allow-query     { localhost; 10.0.10.0/24; };

/*
 - If you are building an AUTHORITATIVE DNS server, do NOT enable recursion.
 - If you are building a RECURSIVE (caching) DNS server, you need to enable
   recursion.
```

```
 - If your recursive DNS server has a public IP address, you MUST enable access
   control to limit queries to your legitimate users. Failing to do so will
   cause your server to become part of large scale DNS amplification
   attacks. Implementing BCP38 within your network would greatly
   reduce such attack surface
*/
recursion yes;

forwarders {
                8.8.8.8;
                8.8.4.4;
        };

dnssec-enable yes;
dnssec-validation yes;

/* Path to ISC DLV key */
bindkeys-file "/etc/named.root.key";

managed-keys-directory "/var/named/dynamic";

pid-file "/run/named/named.pid";
session-keyfile "/run/named/session.key";
};

logging {
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};

zone "." IN {
type hint;
file "named.ca";
};

include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";
include "/etc/named/named.conf.local"; // Include for the file
                                       //where the new zones are stored.
```

named.conf.local

```
    // A record file
zone "okd.local" {
    type master;
    file "/etc/named/zones/db.okd.local"; # zone file path
};

// PTR record file
zone "10.0.10.in-addr.arpa" {
    type master;
    file "/etc/named/zones/db.10.0.10";  # 10.0.10.0/24 subnet
};
```

A records

```
    $TTL    604800
@       IN      SOA     service.okd.local. admin.okd.local.(
                  1     ; Serial
             604800     ; Refresh
              86400     ; Retry
            2419200     ; Expire
             604800     ; Negative Cache TTL
)

; NS records
    IN      NS      service

; A record for the nameserver
service.okd.local.                          IN    A
    10.0.10.10

; A records for machines
; This one can be removed after cluster initialization
bootstrap.lab.okd.local.                     IN    A
    10.0.10.11

; For controll plane nodes
controll-plane-1.lab.okd.local.              IN    A
    10.0.10.12
controll-plane-2.lab.okd.local.              IN    A
    10.0.10.13
controll-plane-3.lab.okd.local.              IN    A
    10.0.10.14

; For worker nodes
compute-1.lab.okd.local.                     IN    A
    10.0.10.15
compute-2.lab.okd.local.                     IN    A
    10.0.10.16

; A records for api's and for all apps created.
```

```
api.lab.okd.local.                                IN    A
    10.0.10.10
api-int.lab.okd.local.                            IN    A
    10.0.10.10
*.apps.lab.okd.local.                             IN    A
    10.0.10.10

; A records for accessing the cluster.
console-openshift-console.apps.lab.okd.local. IN    A
    10.0.10.10
oauth-openshift.apps.lab.okd.local.               IN    A
    10.0.10.10

; A record for etcd, need one for each controll plain node.
etcd-0.lab.okd.local.                             IN    A
    10.0.10.12
etcd-1.lab.okd.local.                             IN    A
    10.0.10.13
etcd-2.lab.okd.local.                             IN    A
    10.0.10.14


; SRV records, need one for each controll plane node.
_etcd-server-ssl._tcp.lab.okd.local.    86400     IN    SRV     0
        10    2380    etcd-0.lab
_etcd-server-ssl._tcp.lab.okd.local.    86400     IN    SRV     0
        10    2380    etcd-1.lab
_etcd-server-ssl._tcp.lab.okd.local.    86400     IN    SRV     0
        10    2380    etcd-2.lab
```

PTR records

```
    $TTL    604800
@       IN      SOA     service.okd.local. admin.okd.local. (
                  6     ; Serial
             604800     ; Refresh
              86400     ; Retry
            2419200     ; Expire
             604800     ; Negative Cache TTL
)

; NS records
    IN      NS      service.okd.local.

; PTR record for the nameserver
10    IN    PTR    service.okd.local.

; PTR records for the different nodes.
11    IN    PTR    bootstrap.lab.okd.local.
12    IN    PTR    controll-plane-1.lab.okd.local.
13    IN    PTR    controll-plane-2.lab.okd.local.
14    IN    PTR    controll-plane-3.lab.okd.local.
```

```
15    IN    PTR    compute-1.lab.okd.local.
16    IN    PTR    compute-2.lab.okd.local.

; PTR record for the api's.
10    IN    PTR    api.lab.okd.local.
10    IN    PTR    api-int.lab.okd.local.
```

D.1.3 DHCP konfigurasjon

```
    # Domain settings
option domain-name     "okd.local";
option domain-name-servers     10.0.10.10;

# Leastime
default-lease-time 600;
max-lease-time 7200;

# this DHCP server to be declared valid
authoritative;

# specify network address and subnetmask
subnet 10.0.10.0 netmask 255.255.255.0 {
  # DHCP config
  range dynamic-bootp 10.0.10.20 10.0.10.30;
  option broadcast-address 10.0.10.255;
  option routers 10.0.10.1;

  # PXE settings
  # Boot-loader file for BIOS boot.
  filename "pxelinux.0";
  # TFTP server address
  next-server 10.0.10.10;

}

# Staticly allocated addresses for kluster nodes.

# Bootstrap, only needed for initial deployment
host bootstrap {
  hardware ethernet 00:15:5D:3C:E3:05;
  fixed-address 10.0.10.11;
}
```

```
# Maseter nodes --> control plane
host controll-plane-1 {
  hardware ethernet 00:15:5D:3C:E3:06;
  fixed-address 10.0.10.12;
}

host controll-plane-2 {
  hardware ethernet 00:15:5D:3C:E3:06;
  fixed-address 10.0.10.13;
}

host controll-plane-3 {
  hardware ethernet 00:15:5D:3C:E3:06;
  fixed-address 10.0.10.14;
}

# Worker nodes
host compute-1 {
  hardware ethernet 00:15:5D:3C:E3:06;
  fixed-address 10.0.10.15;
}

host compute-2 {
  hardware ethernet 00:15:5D:3C:E3:06;
  fixed-address 10.0.10.16;
}
```

D.1.4 PXE konfigurasjon

```
default bootstrap
label bootstrap
        kernel f32/vmlinuz
        append initrd=f32/initrd.img,f32/rootfs coreos.inst.install_dev=/dev/sda
        coreos.inst.ignition_url=http://10.0.10.10:8080/okd/bootstrap.ign
```

D.1.5 OKD installasjons konfigurasjon

```
apiVersion: v1
 # Name of the domain, same as the one spesified in the DNS configuration.
baseDomain: okd.local
metadata:
  name: lab # Name  of the cluster.

# Worker node configuration.
compute:
```

```
- hyperthreading: Enabled
  name: worker
  replicas: 0

# Controll-plane node configuration.
controlPlane:
  hyperthreading: Enabled
  name: master
  replicas: 2

# Internal cluster network
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  networkType: OpenShiftSDN
  serviceNetwork:
  - 172.30.0.0/16

platform:
  none: {}

fips: false

# A official pullsecret can be obtained by creating a RedHat account.
pullSecret: '{"auths":{"fake":{"auth": "bar"}}}'

# Key of the node that can SSH into the cluster nodes.
# This is not strictly needed.
sshKey: 'ssh......'
```

D.2 Konfigurasjonsfiler for OKD kluster

D.2.1 Image registry

```
# Persistent volume for the registry.
# Configured to use a nsf share for storage.
apiVersion: v1
kind: PersistentVolume
metadata:
  name: registry-pv
spec:
  capacity:
```

```
  # Size of the PV. Needs to be atleast 100 GB according to the documentation.
    storage: 100Gi
  accessModes:
    - ReadWriteMany
  # What happends with the data on the PV when it is removed.
  persistentVolumeReclaimPolicy: Retain
  nfs:
  # Path to the nfs shared folder
    path: /mnt/storage
  # Address to the NFS server
    server: 10.0.10.10
```

D.2.2 Dynamisk lagring

D.2.3 Lagrings klasse

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-nfs-storage
provisioner: nfs-storage #Needs to be the same defined in the deployment.yaml file.
parameters:
  archiveOnDelete: "true"
```

Lagrings utrullering

```
    apiVersion: apps/v1
kind: Deployment
metadata:
  name: nfs-client-provisioner
  labels:
    app: nfs-client-provisioner
  # replace with namespace where provisioner is deployed
  namespace: nfs
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: nfs-client-provisioner
  template:
    metadata:
      labels:
        app: nfs-client-provisioner
```

```
    spec:
      serviceAccountName: nfs-client-provisioner
      containers:
        - name: nfs-client-provisioner
          image: k8s.gcr.io/sig-storage/nfs-subdir-external-provisioner:v4.0.2
          volumeMounts:
            - name: nfs-client-root
              mountPath: /persistentvolumes
          env:
            - name: PROVISIONER_NAME
              value: nfs-storage # This name need to
                                 # be put into the class.yaml file.
            - name: NFS_SERVER
              value: 10.0.10.10   # Path to the nfs-server
            - name: NFS_PATH
              value: /var/nfsshare/dynamic  # Folder that is shared
      volumes:
        - name: nfs-client-root
          nfs:
            server: 10.0.10.10  # Path to the nfs-server
            path: /var/nfsshare/dynamic # Folder that is shared
```

Lagrings Rbac konfig

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nfs-client-provisioner
  # replace with namespace where provisioner is deployed
  namespace: nfs
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
```

```
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "update", "patch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: run-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    # replace with namespace where provisioner is deployed
    namespace: nfs
roleRef:
  kind: ClusterRole
  name: nfs-client-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  # replace with namespace where provisioner is deployed
  namespace: nfs
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  # replace with namespace where provisioner is deployed
  namespace: nfs
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    # replace with namespace where provisioner is deployed
    namespace: nfs
```

```
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
  apiGroup: rbac.authorization.k8s.io
```

D.2.4 Konfigurasjon for HTPasswd

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: my_htpasswd_provider
    mappingMethod: claim
    type: HTPasswd
    htpasswd:
      fileData:
       # Refrense to the secret that was made in the earlyer step.
        name: htpass-secret
```

Vedlegg E

Git readme.md

E.1 okd-cluster-with-spark

Table of Contents

Requirements

Setup

OKD Install

Introduction

In this install guide commands used to install, configure and setup support services and OKD will be listed. As part of this git repo, configuration files used is availible, and all nessisary configuration is done prior to the install phase.

Node name	Operating system	RAM (GB)	vCPU	Disk (GB)
Services	Fedora Server	4	2	50
Bootstrap	Fedora CoreOS	16	4	120
Control plane	Fedora CoreOS	16	4	120
Compute	Fedora CoreOS	8	2	120

Minimum requirements for Nodes and Operating system.

Wget and git

In order to download files later wget will be used, and git is used to get the repo with configuration files.

```
sudo dnf install git wget -y
```

DNS

For DNS Bind will be used. The bind-utils package is only needed for DNS testing purposes.

```
sudo dnf install bind bind-utils -y
```

In order to make a clutter in the named.conf file, as litle editing has been done to it. What has been added is:

- Added the service nodes IP-addres in the *listen-on port 53* field.
- Change the *allow-query* to the subnet the cluster is running on.
- Added forwarders
- Include a file where the *DNS zones* are defined.

Copy the named.conf file to its location.

```
sudo cp okd-cluster-with-spark/support_config/named.conf /etc/named.conf
```

Zone files In the zone file, two zones are defined. One with A records, and one for PTR records.

```
sudo cp okd-cluster-with-spark/support_config/named.conf.local /etc/named/named.conf
```

A- and PTR records Create a new directory for the record files, and copy the files to the location.

```
sudo mkdir /var/named/zones
sudo cp okd-cluster-with-spark/support_config/db.okd.local /var/named/zones/
sudo cp okd-cluster-with-spark/support_config/db.10.0.10 /var/named/zones/
```

Enable, start and check the named service.

```
sudo systemctl enable named
sudo systemctl start named
sudo systemctl status named
```

Open firewall port

```
sudo firewall-cmd --permanent --add-port=53/udp
sudo firewall-cmd --reload
```

Test DNS configuraton In order to make sure that the configuration is correct, change the dns settings and restart the NetworkManger.

```
sudo nmcli connection modify eth0 ipv4.dns "127.0.0.1"
sudo systemctl stop NetworkManager
sudo systemctl start NetworkManager
```
The interface can be something else than eth0
Use DIG command to test DNS settings.
```
dig okd.local
dig -x 10.0.10.10
```

Loadbalancer

For loadbalancing HAProxy wil be used.
The cluster requiers two loadbalancer.

1. API loadbalancer. This functions as the common endpoint to interact and configure the OKD plattform.
    - It needs load balancing at Layer 4, which can be referd to as Raw TCP.
    - A stateless load balancing algorithm.
2. Application ingress load balancer
    - It needs load balancing at Layer 4, which can be referd to as Raw TCP.
    - It is recomended to have a connection-based or session-based persistence.

Api loadbalancer This ports has to be configured for front and backend.

Application Ingress Loadbalancer This ports has to be configured in front and backend.

Install HAProxy and copy config file
```
sudo dnf install haproxy -y
sudo cp okd-cluster-with-spark/support_config/haproxy.cfg /etc/haproxy/haproxy.cfg
```

Set SELinux bit This is done inorder to allowe HAProxy to bind port 80 and 443.
```
sudo setsebool -P haproxy_connect_any 1
```

Enable, start and check the service
```
sudo systemctl enable haproxy
sudo systemctl start haproxy
sudo systemctl status haproxy
```

Configure the firewall for HAProxy

```
sudo firewall-cmd --permanent --add-port=6443/tcp
sudo firewall-cmd --permanent --add-port=22623/tcp
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --permanent --add-service=https
sudo firewall-cmd --reload
```

Web server

For the installation, a simple webserver is needed to host the Ignition files Fedora CoreOS requiers. Because of this, a simple apache webserver is used.

Install HTTPd

```
sudo dnf install httpd -y
```

The only thing that is change in the configuration is the port number. Because of this a separat configuration file is note present.
Change the port from 80 to 8080.

```
sudo sed -i 's/Listen 80/Listen 8080/' /etc/httpd/conf/httpd.conf
```

Set SELinux bit As with HAProxy a *SELinux bit* has to be set.

```
sudo setsebool -P httpd_read_user_content 1
```

Enable, start and check HTTPd

```
sudo systemctl enable httpd
sudo systemctl start httpd
```

Configure the firewall for HTTPd

```
sudo firewall-cmd --permanent --add-port=8080/tcp
sudo firewall-cmd --reload
```

The webservers only intent is to give Fedora CoreOS nodes access to configuration files. This means it will be shutdown after the cluster is operational.

PXE boot

Preboot Execution Environment (PXE) boot is a method of booting over a network with the help of the *Syslinux* bootloader. This setup uses a Trivial File Transport Protocol (TFTP) server, where the operating system files is stored.

Install TFTP

```
sudo dnf install tftp-server tftp -y
```

Enable, start and check the service

```
sudo systemctl enable tftp.socket
sudo systemctl start tftp.socket
sudo systemctl status tftp.socket
```

Add firewall rules

```
sudo firewall-cmd --add-service=tftp --perm
sudo firewall-cmd --reload
```

Syslinux

In order to setup the PXE environment, the sylinux package and shim-x64 and grub2-efi-x64 packages needs to be installed.

Install packages

```
sudo dnf install syslinux shim-x64 grub2-efi-x64
                  --installroot=/tmp/fedora --releasever 33 -y
```

Create folder for PXE configuration files and copy needed files

```
sudo mkdir -p /var/lib/tftpboot/pxelinux.cfg
sudo cp /tmp/fedora/usr/share/syslinux/{pxelinux.0,menu.c32,vesamenu.c32,
ldlinux.c32,libcom32.c32,libutil.c32} /var/lib/tftpboot/
```

Create folder for operating system files

```
sudo mkdir -p /var/lib/tftpboot/f32
```

Download operating system files

```
sudo wget https://builds.coreos.fedoraproject.org/prod/streams/testing-devel/builds/
32.20201111.20.0/x86_64/fedora-coreos-32.20201111.20.0-live-kernel-x86_64
-O /var/lib/tftpboot/f32/vmlinuz

sudo wget https://builds.coreos.fedoraproject.org/prod/streams/testing-devel/builds/
32.20201111.20.0/x86_64/fedora-coreos-32.20201111.20.0-live-initramfs.x86_64.img
-O /var/lib/tftpboot/f32/initrd.img

sudo wget https://builds.coreos.fedoraproject.org/prod/streams/testing-devel/builds/
32.20201111.20.0/x86_64/fedora-coreos-32.20201111.20.0-live-rootfs.x86_64.img
-O /var/lib/tftpboot/f32/rootfs
```

PXE configuration files

Inorder to enable boot based on NIC MAC-address, a file named *01-xx-xx-xx-xx-xx-xx* has to be made. The *xx-* is the MAC-address. Letters in the MAC-address has to be in lowercase.

```
sudo cp okd-cluster-with-spark/support_config/01-00-15-5d-3c-e3-05
/var/lib/tftpboot/pxelinux.cfg/01-00-15-5d-3c-e3-05
```
This file is for the bootstrap node. For the master or worker node, change the ignition file spesified to master or worker.

Cluster initialization

Download installer and oc programs, and extract them

Make a new folder to keep the installer and oc programs.
```
mkdir installers
cd installers
wget https://github.com/openshift/okd/releases/download/4.5.0-0.okd-2020-07-29-07031
openshift-client-linux-4.5.0-0.okd-2020-07-29-070316.tar.gz
wgethttps://github.com/openshift/okd/releases/download/4.5.0-0.okd-2020-07-29-070316
openshift-install-linux-4.5.0-0.okd-2020-07-29-070316.tar.gz
```
Extract the files, and remove tar.gz files.
```
tar -xvf openshift-client-linux-4.5.0-0.okd-2020-07-29-070316.tar.gz
tar -xvf openshift-install-linux-4.5.0-0.okd-2020-07-29-070316.tar.gz
rm openshift-client-linux-4.5.0-0.okd-2020-07-29-070316.tar.gz
openshift-install-linux-4.5.0-0.okd-2020-07-29-070316.tar.gz
```

Create manifests and ignition files

In order to create manifests and ingnition files, the *openshift-install* i used. For this installation a yaml configuration file is use, known as *install-config.yaml*. The format for this file is based on the example from OKD documentation. Create a folder where the manifests and ignition files will be createrd, and move the install-config.yaml file there.
```
mkdir install
cp okd-cluster-with-spark/support_config/install-config.yaml ~/install
```

Create manifests Before this step, take a copy of the *install-config.yaml* file. The install program consumes the file in the process.
```
cp ~/install/install-config.yaml ../install/install-config.yaml.bak
```
This will create kubernetes manifest files.
```
./openshift-install create manifests --dir=~/install
```

Create ignition files After the manifests is create, the ignition files used by FCOS can be generated.
```
./openshift-install create ignition-configs --dir=~/install
```

When this is done, copy the ignition files to the web server. A separat folder for the files is create, and to be sure that there are no issues with file ownership and what users can do with it, the owner and permitions are changed.

```
sudo mkdir /var/www/html/okd
cp ~/install/*.ign  /var/www/html/okd/
chmod 777 -R  /var/www/html/
chown apache: -R  /var/www/html/
```

Start the bootstrap and master nodes At this point all suport services and file should be in place, and the bootstrap node can be created and started. Rimember to check that the MAC-address of the node is the same as the one spesifyed in the DHCP configuration, and that the TFTP file name is correct.

During the installation The installation happends in two stepes.
In the firste step, the bootstrap node is present. To follow the general progression of this:

```
./openshift-install wait-for bootstrap-complete --log-level=debug --dir=~/install
```

or log in to the bootstrap node with ssh, there is a command on the loginscreen that can be used to see how far in the process it is.
In the second phase, the bootstrap node is nologer needed and can be turned off. To follow the general progression of this phase:

```
./openshift-install wait-for install-complete --log-level=debug --dir=~/install
```

During both phases, it is posible to see what cluster operators are currently pressent with:

```
export KUBECONFIG=~/install_dir/auth/kubeconfig
./oc get clusteroperators
```

The install process can take upto one houer, and a link will be posted for logging in to the web console with the password of the *kubeadmin* user when it is done.

Post install configuration

In this section, the post installation configuration will shown.

Authentication

HTPasswd As a proof of consept htpasswd has been used for authentication. This is not a good secure solution, and OKD supports many different methods for authentication.
First the httpd-tools package needs to be downloaded.

```
sudo dnf install httpd-tools
```

Two user will be created for use in later steps.
A users.htpasswd file with username and passwords will be created.

```
htpasswd -c -B -b users.htpasswd admin password
htpasswd -B -b useres.htpasswd bobthebuilder wordpass
```
Afther the file has been created a HTPasswd secret is created:
```
./occreate secret generic htpass-secret --from-file=htpasswd=users.htpasswd
-n openshift-config
```
In order to tell OKD to use the htpwasswd file, a htpassword identity provider is created and the applyed to the cluster.
```
./oc apply -f htpasswd.yaml
```
See htpasswd.yaml file for configuration example
When this is done, you should be able to login with the users created.
```
./oc/oc login -u admin
```

Update htpasswd In order to add or remove useres, the htpwasswd file needs to be retrievd from the cluster.
```
 ./oc get secret htpass-secret -ojsonpath={.data.htpasswd} -n openshift-config |
 base64 --decode > users.htpasswd
```
Add a user to the file:
```
htpasswd -B -b useres.htpasswd newuser newpassword
```
Remove a user from the file:
```
htpasswd -D users.htpasswd newuser
```
When the changes is made, the cluster needs to be updated with the new users.htpasswd file.
```
 ./oc create secret generic htpass-secret --from-file=htpasswd=users.htpasswd
 --dry-run=client -o yaml -n openshift-config | ./oc/oc replace -f -
```
The new user wont show up before it has logged inn ones.
If a user was deleted, it also has to be removed from any resoureses it was asosiated with in the cluster:
```
./oc delete user newuser
./oc delete identity my_htpasswd_provider:newuser
```

Add a new cluster admin user

In order to make the admin user, created in the steps about HTPasswd, a *cluster-admin* user, the correct roles has to be set. In order to do this step, a *cluster-admin* user needs to be used.
```
./oc adm policy add-cluster-role-to-user cluster-admin admin
```

Remove the temporary kubeadmin user Before the kubeadmin user gets removed, it is **important** that a new cluster-wide *cluster-admin* user is created. When that is done, the kubeadmin user can be removed with:
```
./oc delete secrets kubeadmin -n kube-system
```

Add a regular user with its own project

First a new project is created
`./oc new-project test-project`
Add the user to the project.
`./oc adm policy add-role-to-user admin bobthebuilder -n test-project`
The admin keyword makes bobthebuilder an admin in the test-project

Add a user to a new group

First a new group is created:
`./oc adm groups new test-group`
The we add a existing user to the group:
`./oc adm groups add-users test-group bobthebuilder`

Storage

For the next two sections storage is the focuse. In order to create a image registry and a dynamic PV, a nfs share is used. The nfsshare is setup with the help of Ryan Hay.

Dependencies

- nfs-utils

Install packages, enable and start the serveices.

```
sudo dnf install -y nfs-utils
sudo systemctl enable nfs-server rpcbind
sudo systemctl start nfs-server rpcbind
```

Next create a folder that will be shared, one for the image registry and one for the dynamic PV, change owner and permitions:

```
sudo mkdir -p /var/nfsshare/
sudo mkdir -p /var/nfsshare/dynamic
sudo chmod -R 777 /var/nfsshare
sudo chown -R nobody:nobody /var/nfsshare
```

Add two entries to the */etc/exports* file:

```
echo '/var/nfsshare 10.0.10.10/24(rw,sync,root_squash,no_all_squash,no_wdelay)' |
sudo tee /etc/exports
echo '/var/nfsshare/dynamic 10.0.10.10/24(rw,sync,root_squash,no_all_squash,no_wdela
| sudo tee /etc/exports
sudo exportfs -rv
```

Change a SELinux bool:
```
sudo setsebool -P nfs_export_all_rw 1
```

Restart the service and add a firewall rule:
```
sudo systemctl restart nfs-server
sudo firewall-cmd --permanent --zone=public --add-service nfs
```
At this point the nfsshare should be ready, and we can proseed with configuring the image registry and the dynamic PV.

Add a image registry

Befor users can create new deployments, a image registy needs to be created. This is the place where images is stored.
A persistent volume (pv) has to be created. How this pv looks is spesified in the file *registry_pv.yaml*.
```
./oc apply -f okd-cluster-with-spark/support_config/registry_pv.yaml
```
After this a change to the registry configuration has to be made.
```
./oc/oc edit configs.imageregistry.operator.openshift.io
```
In this file a *managementStat*, and *storage* filed needs to be changed.
```
managementStat: Removed --> managementStat: Managed
storage: {}             --> storage:
                              pvc:
                                claim:
```
By setting the *managementStat* from *Removed* to *Managed* allows for building and pushing of images to the registry, and making the *claim* empty in the storage section allows the automatic creation of an *image-registry-storage* persistent volume clamin (PVC).
In order to see if the pv was created successfully use:
```
./oc get pv
```

Add dynamic persistent storage

The files used for configuring dynamic storaged comes from Kubernetes-sigs.
For the dynamic storage, tre components is needed.

1. A storage class
2. A deployment
3. A rbac configuration.

Create a new project for the dynamic storage
```
./oc new-project nfs
```

Add a storage class
```
./oc create -f okd-cluster-with-spark/storage_config/dynamic_class.yaml
```

Add a deployment
`./oc create -f okd-cluster-with-spark/storage_config/dynamic_deployment.yaml`

Add the Rbac
`./oc create -f okd-cluster-with-spark/storage_config/dynamic_rbac.yaml`
In order to make the deplyment go throug, a new role has to be created and given to the service account for the *nfs* project

Create new role:
```
./occreate role use-scc-hostmount-anyuid --verb=use --resource=scc
--resource-name=hostmount-anyuid -n nfs
```

Find the name of the service account:
`./oc get sa`

Add it to the service account:
```
./ocadm policy add-role-to-user use-scc-hostmount-anyuid -z nfs-client-provisioner
--role-namespace nfs -n nfs
```

Make the dynamic pv the default storage method
`./oc edit sc managed-nfs-storage`
Need to edit the deployment by adding:
```
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
```

Check that the new storage class (sc) is the default one
`./oc get sc`
Should show:
```
NAME                          PROVISIONER RECLAIMPOLICY   VOLUMEBINDINGMODE
managed-nfs-storage (default) nfs-storage Delete          Immediate
```

Add a new node to an existing cluster

In order to add new nodes to the cluster, a valid sertificate needs to be present. This example updates the *worker* ignition file. In order to add a master node, change the output to *~/install/master.ignition*

Update current worker ignition with correct certificate Dependency:

- openssl

Make sure that the dependancey is present.
`sudo dnf install openssl -y`

Update with the correct cluster name and domain name.
The command comes from a exelent blog by Filipe Miranda on how to add new nodes.

```
MCS="api-int.lab.okd.local:22623"NEWBASE64CERT=$(echo "q" |
openssl s_client -connect $MCS  -showcerts 2>/dev/null | awk
'/-----BEGIN CERTIFICATE-----/,/-----END CERTIFICATE-----/' | base64 --wrap=0)
&& sed --regexp-extended --in-place=.backup
"s%base64,[^,]+%base64,$NEWBASE64CERT\"%" ~/install/worker.ign
```

When the ignition file is updated, copy it to the webserver, and change it't permitions.

```
sudo cp ~/install/*.ign  /var/www/html/okd/
sudo chmod 777 -R  /var/www/html/
sudo chown apache: -R  /var/www/html/
```

Sign sertificate request This needs to be don in order for the node to be added to the cluster, and it has to be done twise. The command accepts all pending sertificate requests.

```
./oc get csr | grep Pending | awk '{print $1}' | xargs ./oc adm certificate approve
```

After this is done, a new node should be seen in the web-console.

Apache HDFS

Can be reccomended to look through this tutorial as well to see how the hadoop processes can be seperated with making a new user and group only for hadoop. To setup a HDFS pseudo-distributed cluster on a single-node where each Hadoop deamon runs in a seperate Java process. Following the official documentation with additional information. This is set up on a Fedora 33 (5.8.15-301.fc33.x86_64) cloud base image in Openstack with the following whitch is what we recomend as a minimum since having less resources seem to crash the service.

RAM (GB)	vCPU	Disk (GB)
4	2	40

Required software: Java, list of supported [versions:]
(https://cwiki.apache.org/confluence/display/HADOOP/Hadoop+Java+Version)
We ended up using OpenJDK 8 from Openlogic.com after testing the Java 8/11 from the package manager.
The node also needs to have ssh and pdsh installed, this can be done by the local package manager.

```
sudo dnf install ssh
sudo dnf install pdsh
```

Download Hadoop also linked at hompage

Prepare to start cluster

```
 //set to the root of your Java installation
export JAVA_HOME=/usr/java/latest
```
By running the following command can you check if java is working as it should.
`bin/hadoop`

Configuration

Following config is from the official[link] documentation:
etc/hadoop/core-site.xml:
```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```
etc/hadoop/hdfs-site.xml:
```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```
In order to use full storage space of the node used is there need to make a new directory for HDFS to use and edit the configuration. Answer found in this forum post. example.
`mkdir /home/myUserID/hdfs`
Then add this to etc/core-site.xml:
```
   <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/myUserID/hdfs</value>
    <description>base location for other hdfs directories.</description>
   </property>
```

Passphraseless ssh

Check if possible to access localhost whithout password:
`ssh localhost`
If that don't work:
```
  $ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
  $ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
  $ chmod 0600 ~/.ssh/authorized_keys
```
If it still don't work and the error is "Connection refused" a quickfix to the config file $HADOOP_HOME/libexec/hadoop-functions.sh can fix the error.

It is a known error in HDFS with the use of rsh in pdsh. Answer jira case on the error HADOOP-15219
Edit the file $HADOOP_HOME/libexec/hadoop-functions.sh Quickfix 1 from:
```
if [[ -e '/usr/bin/pdsh' ]]; then
```
To:
```
if [[ ! -e '/usr/bin/pdsh' ]]; then
```
Quickfix 2 From:
```
PDSH_SSH_ARGS_APPEND="${HADOOP_SSH_OPTS}" pdsh \
```
To:
```
PDSH_RCMD_TYPE=ssh PDSH_SSH_ARGS_APPEND="${HADOOP_SSH_OPTS}" pdsh \
```

Start cluster

After finishing editing configurations for HDFS format the filesystem:
`bin/hdfs namenode -format`
To start NameNode daemon and DataNode daemon:
`sbin/start-dfs.sh`
or all with:
`sbin/start-all.sh`

Troubleshooting

In order to edit the configuration of HDFS after it is formatted and set up do the files in hdfs may need to be deleted. first stop all nodes
`sbin/stop-all.sh`
Then delete all files in the folder hdfs uses, e.g /home/user/hdfs
`rm -rf /home/user/hdfs`
Then reformat the node with:
`bin/hdfs namenode -format`
Then restart the komponents:
`sbin/start-all.sh`
hadoop [https://www.apache.org/dyn/closer.cgi/hadoop/common/]

Overview of commands in HDFS

Blogpost listing many of the commands

Administrative

User and groups OKD official documentation

Users There can be made different user with different cluster roles and local roles based on needs for access.
To create the user : `oc create user username`
// Dette er ikke helt klart hva som kreves via cli, enklere å gjøre via nettleser.

Developer

Projects In OKD is it usual to divide applications into projects to further improve structure. To list all the projects use the command `oc get projects`
To see what project that is currently beeing worked on with the command `oc project`
To create a new project the command `oc new-project <projectName>`
To create a project with more information attached can the following two flags be appended to the command. `--description="<description>"` and `--display-name="<display_name>"`
To check the status of the project use `oc status` To delete a project can the following command be used. `oc delete project <projectName>`

Application To read more about making a project from CLI and supported function is this link to the official documentation.
https://docs.okd.io/latest/applications/application_life_cycle_management/creating-applications-using-cli.html
`oc new-app <baseImage> <git-repo/source code>` It is also possible to add a new application with a docker image directly from ex. Docker hub
oc new-app http,gitrepo.git# // her kan man bestemme hvilken branch som skal brukes.
To be able to view the new created application is the command `oc expose <appName>` to expose the application to the outside. Then with the command `oc get route <name>` to discover the link to the application. Without the *name* parameter in the last command will the
To list pods/applications running on the current project run the command `oc get pods`. This will show a list including the age of the pod and
To delete an application run the command `oc delete pod <Name>`
///////// Build strategies: Docker build Må lage egen dockerfil
s2i -https://www.youtube.com/watch?v=-RqiHB-bnkg
https://github.com/openshift/source-to-image // git-repo S2I https://notes.elmiko.dev/2019/06/21/s2i-gitlab-ci.html // gitlab S2I
Python base-image kan hentes fra docker hub. Java eller ruby på bruke oppenshoft registry

Jupyterlab

These steps are taken from: https://zero-to-jupyterhub.readthedocs.io/en/stable/jupyterhub/installation.html
Generate a random 32 bit hex string to be used as a security token.

```
openssl rand -hex 32
```
Take the security token and add it to a file named *config.yaml.*
```
nano config.yaml
---
## This will be added to config.yaml
## <RANDOM_HEX> is generated by openssl rand -hex 32
proxy:
   secretToken: "<RANDOM_HEX>"
singleuser:
   image:
      name: jupyter/all-spark-notebook
      tag: 584f43f06586
```
Add the helm repository for jupyterhub, and update the helm repository.
```
helm repo add jupyterhub https://jupyterhub.github.io/helm-chart/
helm repo update
```
Now we can install the helm start with the config.yaml file created earlier. This command is done in the same folder as the config.ymal file is located.
```
helm upgrade --cleanup-on-fail \
--install jhub jupyterhub/jupyterhub--namespace default \
--create-namespace \
--version=0.10.6 \
--values config.yaml
```
This takes a while. Add a route to the pod named *hub* in OKD. When this is done, you should be able to log into the jupyterhub running on the OKD cluster.

Downloading Apache Spark

Apache spark can be downloaded from https://spark.apache.org/downloads.html or from the official apache spark github repository at https://github.com/apache/spark This section show how to download spark 3.1.1 pre built for hadoop 3.2 and later from sparks website using CLI.
```
wget https://downloads.apache.org/spark/spark-3.1.1/spark-3.1.1-bin-hadoop3.2.tgz
tar xvf spark-3.1.1-bin-hadoop3.2.tgz
```
After downloading and extracting, change diretory to spark-3.1.1-bin-hadoop3.2 where all further commands regarding apache spark will be run from
```
cd spark-3.1.1-bin-hadoop3.2
```

Building a Spark Docker image

Requirements:
Downloading Apache Spark
Apache Spark comes prebundled with a tool for building dockerfiles for apache spark. The tool is located in the bin directory of the installation.
To build a basic dockerimage for spark run the following command:

```
./bin/docker-image-tool.sh -t basic build
```
Building a docker image with python support
```
./bin/docker-image-tool.sh  -t pythonsupport -p
kubernetes/dockerfiles/spark/bindings/python/Dockerfile build
```

Setting up a service account for apache spark

Requirements:
Authenticate to the OKD cluster
OKD uses RBAC and therefore a service account is needed to access the API server in OKD
```
oc create serviceaccount spark
oc create clusterrolebinding spark-role
--clusterrole=edit --serviceaccount=default:spark --namespace=default
```

Installing GCP spark operator

Requirements:
Authenticate to the OKD cluster
Installing helm
```
# curl -L https://mirror.openshift.com/pub/openshift-v4/clients/helm/
latest/helm-linux-amd64 -o /usr/local/bin/helm
# chmod +x /usr/local/bin/helm
```
Installing GCP spark operator
```
$ helm repo add spark-operator https://googlecloudplatform.github.io/spark-on-k8s-op
$ helm install my-release spark-operator/spark-operator --namespace spark-operator
--create-namespace --set webhook.enable=true --set webhook.port=443
```

Using GCP spark operator to submit spark applications

Requirements:
Authenticate to the OKD cluster
Setting up a service account for apache spark
Building a Spark Docker image
Installing GCP spark operator

Using spark-submit to submit spark applications

Requirements:
Authenticate to the OKD cluster
Setting up a service account for apache spark
Building a Spark Docker image
Spark-submit is located in the bin/ directory of the apache spark installation
Minimum arguments:
spark-submit
–master k8s://“Cluster ip address”

–deploy-mode cluster
–name “name of application”
–class org.apache.spark.examples.“Class name”
–conf spark.kubernetes.authenticate.driver.serviceAccountName=“service account”
–conf spark.kubernetes.container.image="“Link to dockerfile”"
“Where you can access the workload”
Example application:
spark-submit –master k8s://https://example.com:6443 –deploy-mode cluster –name spark-pi –class org.apache.spark.examples.SparkPi –conf spark.kubernetes.authenticate.driver.serviceAccountName=spark –conf spark.kubernetes.container.image=“docker.io/anderwm/spark:hostfil” http://exampledomain.com/workload.py