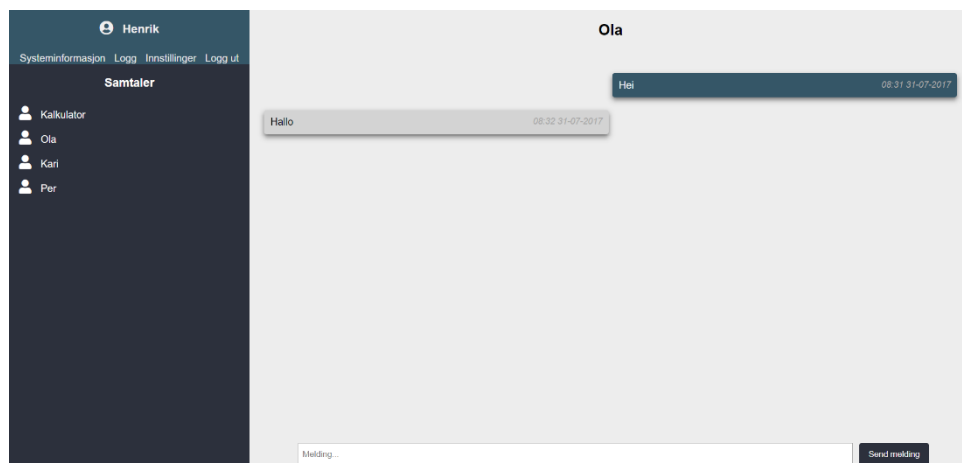


# Oppgaver dag 1

## Intro

Dere har blitt ansatt hos et firma som skal videreutvikle en meldingsapplikasjon de har laget for 2 år siden. Meldingsapplikasjonen er laget for bruk internt i firmaet slik at en kan oppnå enkel kommunikasjon mellom ansatte. På grunn av interne problemer i firmaet har ikke applikasjonen vært i bruk siden den ble laget. Dere skal nå videreutvikle og drifte applikasjonen. Det er viktig for firmaet at applikasjonen er oppe og kjører så snart som mulig.

Meldingsapplikasjonen består av en innloggingsside og en hovedside hvor man kan sende meldinger til hverandre. I tillegg til å sende meldinger mellom ansatte er applikasjonen utstyrt med en bruker som heter kalkulator, som utfører enkle regneoperasjoner. Det er også lagt inn en side for å endre brukerinnstillinger, men denne fungerer ikke.



Produkteier ønsker at første iterasjon skal innebære å få systemet oppe å kjøre, samt videreutvikle kalkulatoren da denne er svært viktig for de ansatte. Det er på forhånd laget user stories og backlog for første iterasjon som dere skal bruke. Det poengteres at dette er et gammelt prosjekt man ikke har full oversikt over. Dette kan føre til uventede problemer man må ta hånd om underveis, og oppdatere backlog.

User stories	Backlog
<p><i>Høy prioritet</i></p> <p>Som utvikler av systemet</p> <p>Ønsker jeg å ha Continuous Integration og Continuous Delivery</p> <p>Slik at nye endringer blir automatisk rullet ut</p>	<ul style="list-style-type: none"> <li>• Installere og starte Jenkins server</li> <li>• Koble GIT til Jenkins server med webhooks for CI/CD</li> </ul>
<p><i>Middels prioritet</i></p> <p>Som bruker av systemet</p> <p>Ønsker jeg å utføre flere regneoperasjoner mot kalkulatoren</p> <p>Slik at jeg kan utføre nødvendige utregninger</p>	<ul style="list-style-type: none"> <li>• Lag test og metode for multiplikasjon på kalkulator</li> <li>• Lag test og metode for divisjon på kalkulator</li> </ul>
<p><i>Høy prioritet</i></p> <p>Som drifter av systemet</p> <p>Ønsker jeg å overvåke ressursbruk av systemet</p> <p>Slik at jeg kan forhindre uønskede situasjoner</p>	<ul style="list-style-type: none"> <li>• Installere Glances for ressursovervåking</li> <li>• Legge til Glances web UI til applikasjonen</li> </ul>
<p><i>Høy prioritet</i></p> <p>Som drifter av systemet</p> <p>Ønsker jeg ha et loggesystem for ressursbruk</p> <p>Slik at jeg ikke trenger å overvåke systemet i nåtid</p>	<ul style="list-style-type: none"> <li>• Sette opp et loggesystem for enklere ressursovervåking</li> <li>• Legge til Grafana verktøy til applikasjonen</li> </ul>

## Oppgave 1

Det første dere må gjøre er å lagre prosjektet på lokal maskin.

- a) En på gruppen forker prosjektet (<https://gitlab.stud.idi.ntnu.no/surya/devops-workshop>) til egen GIT, og gir de andre medlemmene tilgang.
- b) Opprett Access Token for prosjektet. Se vedlegget Gitlab Access Tokens
- c) Klon prosjektet på lokal maskin fra gruppens eget repo.

## Oppgave 2

For at utviklerne skal kunne kjøre prosjektet lokalt på egen maskin trenger de tilgang til 2 databaser. En til kjøring av selve applikasjonen, og en til kjøring av tester. Dere skal derfor sette opp 2 databaser på VMen dere har fått tildelt.

- Koble til VM
- Clone prosjektet til VM
- Databasene skal settes opp med docker. I prosjektet ligger det en fil, docker-compose-local-db.yaml. Dette er en oppskrift på hvordan docker skal sette opp databasene. For å starte databasene med det gitte oppsettet kjøres følgende kommando i prosjektmappen:  

```
$ docker-compose -f docker-compose-local-db.yaml up -d
```
- Denne vil sette opp følgende ting:
  - Database til lokal kjøring
  - Database til lokal testing
  - Databaseklient som brukes til å behandle databasene
- Oppsettet av databasene vil ta litt tid. Når det er ferdig dere logge inn på databaseklienten på [http://ip\\_til\\_vm:8070](http://ip_til_vm:8070). Her er innloggingen følgende (dersom du får en pop-up om at passordet sendes over en ukryptert forbindelse, søk på nett om hvordan man kan fjerne dette!):
  - Server: local-db Server: local-test-db
  - Brukernavn: root Brukernavn: root
  - Passord: example Passord: example
- Sjekk at begge databasene er oppe og kjører, med databasen myChat og tabellene groupChat, message, user, user\_groupChat

## Oppgave 3

Da produkteier ønsker applikasjonen oppe å kjøre er det på tide å få prosjektet på virtuell maskin og sette opp server. Dere skal bruke Jenkins som er en server brukt for automatisk testing og utrulling av prosjektet.

- a) Koble til VM, installer og kjør Jenkins server. Se vedlegg “Installere og kjøre Jenkins server”.
- b) Dere skal nå konfigurere Jenkins server. Dette innebærer å få Jenkins til å kunne kjøre prosjektet og tester, samt koble Jenkins mot GIT. Se vedlegg “Konfigurere Jenkins server” for hvordan dette skal gjøres.

## Oppgave 4

For å videreutvikle et gammelt prosjekt er det viktig å forstå det som tidligere er laget. Les README.md filen skrevet av tidligere utviklere. Denne filen er laget med hensikt at nye utviklere kan lese og forstå systemet. Bruk gjerne denne filen senere dersom noe er uklart.

## Oppgave 5

Det viser seg at prosjektet ikke går i fase “Build” og “Deploy” da en eller flere tester feiler. Da dere har satt opp webhooks til GIT vil de ulike fasene kjøres hver gang dere pusher endringer på branch master. Dersom en av fasene feiler vil ikke de neste fasene bli gjennomført, og de nye endringene vil ikke bli rullet ut. Dere må nå finne årsaken til at prosjektet ikke rulles ut, rette opp i feilen(e) og lagre dette på branch master slik at prosjektet rulles ut.

- a) I web UI til Jenkins server (port 8080) under pipeline kan dere se hvilke faser som feilet. Finn årsaken til at prosjektet ikke rulles ut.
- b) Rett opp i feilen og push dette til branch master. Dersom alt er som det skal, kan dere nå åpne applikasjonen på <http://ip-til-vm>

## Oppgave 6

Applikasjonen skal nå være oppe å kjøre. Da produktet ikke har vært i bruk på lang tid, er det stor usikkerhet rundt ressursbruken av systemet. Produteier ønsker at dere skal se på dette.

- a) For overvåking av ressursbruk skal dere bruke verktøyet Glances. Under ligger kommandoer for hvordan man installerer Glances på VM.

```
$ sudo apt install python3-pip
```

```
$ sudo pip install glances
```

```
$ cd /usr/bin
```

```
$ sudo ln -s /usr/local/bin/glances glances
```

```
$ sudo pip install bottle
```

Kjør Glances med kommando:

```
$ glances -w -q &
```

(Tip: It might display message that 'Glances Web User Interface started on ...'.

If so, you can use ctrl+C, to back to the console in VM)

- b) Produsenter ønsker at overvåking av ressursbruk skal være tilgjengelig fra applikasjonen. Dere skal derfor gjøre at knappen Ressursovervåking navigerer til Glances web UI. Dette gjøres i app.html. A-elementet med tekst "Ressursovervåking" lager en hyperlenke til Glances web UI. Endre href="#" til href="http://ip-til-vm:61208/".
- c) Verktøyet Glances viser ressursbruken av systemet i nåtid. Dere må derfor *bruke* systemet samtidig som dere ser på Glances slik at dere kan se om det er tilfeller av uvanlig ressursbruk. Dette er viktig for dere da dere må ha en forståelse for hvor mye ressurser applikasjonen trenger. Ser dere noe som er merkelig bør dere finne årsak og vurdere eventuelle tiltak som bør gjøres.

## Oppgave 7

Kalkulatoren skal nå videreutvikles slik at den kan utføre flere operasjoner. Det er ønskelig fra produkteier at den skal kunne utføre multiplikasjon og divisjon.

- Lag tester for de nye metodene slik du ønsker de skal fungere. Testene skal ligge i CalculatorResourceTest.java. Husk å oppdatere testCalculate() slik at denne tester for multiplikasjon og divisjon.
- Lag metodene for multiplikasjon og divisjon. Metode hodet for metodene skal se slik ut:  

```
private int multiplication(String equation)
private int division(String equation)
```
- Push endringene til branch master og bekreft at de nye metodene rulles ut.

## Oppgave 8

Da dere overvåket ressursene i oppgave 5 måtte dere følge med i Glances for å fange opp ressursbruken til enhver tid. Dere har ikke mulighet til å følge med i Glances til enhver tid for å fange opp uventet ressursbruk, og må dermed lage et loggesystem slik at dere slipper å overvåke systemet i nåtid.

- a) Loggesystemet innebærer at Glances må lagre informasjonen om ressursbruk. Glances kommer med støtte for å skrive til databasen InfluxDB. Etter at informasjonen er logget til en database trenger vi en måte å lese den på. Dette gjør vi med programmet Grafana. Det er laget to script som:
  - a. Setter opp InfluxDB database
  - b. Setter opp Grafana
  - c. Starter Glances i loggmodus

Først og fremst er vi nødt til å installere en klient for databasen influxdb:

```
$ sudo pip install --system influxdb-client
```

Deretter, gå til prosjekt mappe og gjør fila «setupLogging» eksekverbar som vist tidligere og kjør den med

```
$ ./setupLogging
```

Etter at logging er satt opp (dvs. vi har kjørt opp containere med influxdb og grafana), er vi nødt til å konfigurere glances til å legge data inn i databasen (influxdb) vi skal bruke til å ta vare på systemmetrikkene som glances gir oss. For å få til dette må vi editere fila

glances.conf . Gå til segmentet som er kalt «[influxdb2]» (legg merke til to-tallet på slutten der) under «Exports»-delen av fila. For «host» legg inn IP til Vmen (merk, det går *ikke* an å bruke «localhost» her).

(Husk at endringer ovenfor kan gjøres via vanlig merge request til master og deretter kjør git pull i VM)

Gjør fila «runLogging» eksekverbar og kjør med

```
$ sudo ./runLogging
```

Kommandoene skal kjøres fra rotkatalogen til prosjektet. Se gjennom scriptene for å forstå hva som skjer. Dersom alt fungerer som det skal vil Glances nå kjøre i bakgrunnen og skrive til databasen.

- b) Neste steg er å konfigurere Grafana, følg instruksene som ligger i vedlegget “Konfigurere Grafana”.
- c) Produkteier ønsker at Grafana verktøyet skal være tilgjengelig fra applikasjonen på lik måte som Glances. Knappen “Logg” skal brukes, og linken til Grafana er:  
<http://ip-til-vm:3000/d/ESYAe0tnk>  
Push endringene på branch master, og test at dere kan åpne Grafana fra applikasjonen. Gjør dere kjent med Grafana og hva som logges.

## Oppgave 9

Produkteier har testet kalkulatoren og har kommet med noen tilbakemeldinger. Han mener det er nødvendig å kunne skrive flere enn to tall og at dette er av høy prioritet. I tillegg synes han at responsen ved feil, for eksempel når det sendes inn regnestykker på feil format, er misvisende og at dette må forbedres. I dag er det slik at kalkulatoren svarer med -1 dersom det skjer en feil.

- a) Implementer mulighet for å skrive flere enn to tall. For eksempel skal man ha muligheten for å skrive 1+2+3. Husk å oppdatere testene.

Hint: regulære uttrykk for å takle mer enn to tall er som følger (se i metoden `calculate()`):

Addisjon: `^\d+(?:\s*\+\s*\d+)*$`

Substraksjon: `^\d+(?:\s*\-\s*\d+)*$`

Multiplikasjon:  $^{\wedge} \backslash d+(?: \backslash s^{*}[*] \backslash s^{*} \backslash d+)^{*} \$$

Divisjon:  $^{\wedge} \backslash d+(?: \backslash s^{*}[/] \backslash s^{*} \backslash d+)^{*} \$$

- b) Forbedre respons fra kalkulator. Dette kan gjøres i CalculatorResource.java. Husk å oppdatere testene.

## Oppgave 10

Dere skal nå utføre en retrospektiv, og reflektere over denne iterasjonen. Under er det gitt punkter som dere skal drøfte og diskutere sammen med resten av gruppen. Retrospektiven utføres normalt sett ved at man skriver svarene på gullapper som dere går gjennom i fellesskap etterpå, men i disse pandemiske tider er ikke det mulig. Dere kan selv finne en digital variant, eller bruke <https://www.retrospected.com/> som tillater en å opprette en anonym bruker og tilpasse punktene for retrospektiven. Plukk ut de fire-fem punktene i hver kategori, og fyll dette inn i malen for å dokumentere retrospektiven. Send dette på mail til [surya.b.kathyat@ntnu.no](mailto:surya.b.kathyat@ntnu.no) med «Retrospektiv dag 1, gruppe nr. XX» som subject.

- Hvis vi kunne gjort iterasjonen på nytt:
  - Hva ville vi gjort annerledes?
  - Hva ville vi gjort likt?
- Hva gikk bra?
- Hva kan forbedres?



# Oppgaver dag 2

## Intro

En testgruppe har testet systemet etter forrige iterasjon. De har kommet med flere tilbakemeldinger:

- De mener at ansatte ikke vil bruke systemet da det ikke er passord ved innlogging, noe som gjør at andre kan logge inn på deres bruker.
- Det var personer i testgruppen som ønsket å endre brukernavn, og måtte lage nye brukere da de ikke kunne endre dette.
- Testgruppen ser også at det kan være behov for å implementere gruppesamtaler, da noen meldinger bør kunne sendes til flere enn én ansatt

Produkteier har brukt tilbakemeldingene til å lage user stories. Ut fra de skal dere lage en backlog for hva som må gjøres under neste iterasjon. User story er markert med grad av prioritet.

User story
<i>Høy prioritet</i> Som bruker av systemet Ønsker jeg å logge inn med passord Slik at ingen andre kan bruke brukeren min
<i>Høy prioritet</i> Som bruker av systemet Ønsker jeg å endre brukerinnstillinger Slik at jeg kan tilpasse brukeren min
<i>Lav prioritet</i> Som bruker av systemet Ønsker jeg å opprette gruppesamtaler Slik at jeg kan sende meldinger til flere samtidig

## Oppgave 1

Lag backlog for denne iterasjonen med oppgaver fra gitte user stories. Husk at en user story kan deles opp i mange oppgaver, og at backlog kan utvides i løpet av iterasjonen.

## Oppgave 2

I forrige iterasjon utviklet dere et log-system for ressursbruk. Det er nå naturlig å ta en titt på dette for å se at alt er som det skal. Se om det er noe unaturlig dere må ta tak i.

## Oppgave 3

Dere skal nå implementere passord for pålogging. Dette innebærer også å lage salt og hash-funksjon, slik at passordet blir kryptert. Under er det gitt tips for hvordan dere kan gjennomføre oppgaven, men her må dere også bruke google for å forstå hvordan metodene dere skal bruke fungerer.

- Det aller første dere trenger er et input-felt på innloggingssiden som tar inn et passord. Dette legges til i index.html. Se etter input-felt for brukernavn for tips til hvor dette skal ligge. Input-feltet skal ha attributtene:
  - class="loginInput" og
  - id="password"

Det mangler et attributt som gjør at når man skriver inn passordet blir det skjult med prikker. Finn hvilket attributt dette er ved hjelp av google.

- Når knappen "Logg inn" trykkes kalles funksjonen login(event) i index.js. Informasjonen som sendes til server inneholder kun "username". Legg til passord.

Metodene som beskrives i punkt 3 og 4 finner du i UserDao.java, og testene skal skrives i UserDaoTest.java.

- Metodehodet for å genere salt er laget, men funksjonen returnerer kun en tom bytetabell. Bruk `java.security.SecureRandom` for å generere en bytetabell med lengde 16. Testen for metoden kan sjekke at to genererte salt ikke er like.
- Metodehodet for å hashe passord er laget, men funksjonen er ikke skrevet. Bruk `java.security.MessageDigest` til å hashe passord med saltet laget fra punkt 3. Gjør en vurdering på hvilken algoritme som skal brukes. Bruk google for å se hvilke som er sikker og ikke. Vedlagt ligger kode for å konvertere en byte array til String. Testen(e) for metoden kan blant annet sjekke:
  - Likt passord med likt salt blir lik hash

- Ulikt passord med likt salt blir ulik hash
- Likt passord med ulik salt blir ulik hash

Konvertering av byte array til String:

```
StringBuilder stringBuilder = new StringBuilder();
for(int i = 0; i < bytes.length; i++){
    stringBuilder.append(Integer.toString((bytes[i] & 0xff) + 0x100,
16).substring(1));
}
String hashedPassword = stringBuilder.toString();
```

## Oppgave 4

Det er et krav om at systemet skal kunne brukes av 100 personer samtidig. Vi skal nå simulere at systemet er tatt i bruk av de ansatte. For å gjøre dette bruker vi Gatling load test. Først må Gatling installeres med scriptet som ligger i prosjektet. Gjør filen kjørbart, og deretter kan den kjøres med følgende kommando i rotkatalogen:

```
$ ./installGatling
```

Deretter kan selve testen simuleringen med scriptet runGatling i rotkatalogen:

```
$ ./runGatling
```

Velg 'Run the Simulation locally' og deretter velg simuleringen chat.ChatSimulation ved å skrive inn tallet 0 (det kan ta en god stund før spørsmålet om å velge simulering dukker opp).

Trykk Enter 1 gang for å gi test rapporten standard navn/beskrivelse.

I filen ChatSimulation.scala som ligger i mappen gatling/chat/ kan man endre antallet simulerte brukere. Dette gjøres på kodelinje 74. Ved mange brukere kan et systems ressursbruk øke kraftig. Observer hva som skjer med ressursbruken og hva dere må gjøre for å forbedre. Greier dere å finne ca. brukerantall før systemet begynner å få problemer med å behandle antall requester?

Når load test er kjørt blir det generert en rapport. Path til rapporten kommer opp i terminalvinduet, men denne må lastes ned til egen PC, siden man ikke har noen form for GUI på VMen. For å laste ned hele katalogen med rapporten, kjør følgende:

```
$ sftp -r stud@IP_TIL_VM:path-til-rapport .
```

Eksempel (ikke inkludere 'index.html' visst som path i terminalvinduet):

```
sftp -r stud@IP_TIL_VM:///home/stud/gatling-charts-highcharts-bundle-3.9.1/results/chatsimulation-<somelongnummer>/ .
```

Merk punktumet på slutten av denne kommandoen, dette er ment å være med, og vil laste ned rapporten til den katalogen du står i når du kjører kommandoen.

I rapporten betyr «OK» at en request ble ekspedert/gikk bra, mens «KO» betyr at requesten ble av en eller annen grunn forkastet/ikke ekspedert. Dette kan være av forskjellige grunner, men mest sannsynlig er det pga. for høyt trykk til at serveren greier å behandle alle innkomne requester (det er jo stresstesting vi driver med her).

## Oppgave 5

Implementer funksjonen for å endre brukerinnstillinger hos en bruker. HTML siden er ferdig implementert, men dere må gjøre endringer i tilhørende javascript fil og lage endepunkt på server.

- Dere må først fullføre funksjonen `editUser(event)` i `settings.js`. Slik koden er nå sendes det et tomt objekt til server når man trykker "Lagre". Dere må hente informasjon fra input elementene, for så å lagre dette i JSON-objektet "newInformation". ID til input-feltene dere skal hente informasjonen fra finner dere i `settings.html`. Objektet skal bestå av "username" og "password". Dette gjøres på lik måte som ved `login()` funksjonen dere har jobbet med tidligere. Husk å oppdatere brukernavnet som ligger i `sessionStorage` dersom endringen blir gjort, dette sees ved at responsen fra server er "true". Se hvordan dette gjøres i `index.js`, linje 42.
- Test for det nye endepunktet dere skal lage er kommentert ut i `UserResourceTest.java`. Før dere starter med å lage det nye endepunktet skal dere gjøre så denne ikke er kommentert lengre, slik at dere får testet endepunktet som lages.
- Kallet som gjøres til server er en PUT-request på endepunktet `/api/user/{userId}`. Dette endepunktet på server skal dere legge til i `UserResource.java`.

Nedenfor ser dere metodehodet:

```
public boolean editUser(@PathParam("userId") int userId, User user){}
```

*Hint til oppgaven*

- Alle endepunkter i UserResource.java starter på /user. Dere må definere at dette spesifikke endepunktet ender på /{userId}.
- Endepunktet bruker JSON objekt i request til å lage et User objekt
- Endepunktet sender response med Boolean som JSON
- Endepunktet kaller på metoden editUser i UserDao.java

## Oppgave 6 (optional)

Det kan være naturlig å automatisere sjekk av uventet ressursbruk. Dette kan gjøres ved hjelp av et automatisk varslingssystem, som varsler når for eksempel CPU-bruk overstiger 90%.

Glances, som allerede brukes til ressursovervåking og logging, har også funksjonalitet for varsling. Les om Glances Actions. Et forslag ved høy CPU bruk er for eksempel å skrive til en fil. Dersom dere har ekstra tid, kan dere prøve å sette opp mer avansert varsling. Dette kan for eksempel være sending av mail.

I filen glances.conf under [cpu] kan man se at det er blitt definert user\_critical=90. Glances Actions er å legge til en ny linje med user\_critical\_action=. Dette betyr at når CPU overstiger 90% utføres kommando beskrevet etter likhetstegnet. Merk at Glances må restartes etter endringer er gjort (drep glances-prosessen og kjør runLogging på nytt).

```
pgrep glances
sudo kill <pid>
glances -w -q &
sudo ./runLogging
```

## Oppgave 7

Dere har nå gjort de prioriterte oppgavene gitt fra produkteier, og kan starte med å utvikle gruppesamtaler om dere har tid til overs.

- For å jobbe med gruppesamtaler trenger vi et objekt for gruppesamtaler. Dere skal først lage klassen GroupChat, som inneholder følgende objektvariabler:

```
private int groupChatId;
private String groupChatName;
private ArrayList<Message> messageList = new ArrayList<>();
private ArrayList<User> userList = new ArrayList<>();
```

I tillegg skal klassen ha:

- En tom konstruktør
- Konstruktør som tar inn `groupId` og `groupName` som argument
- Get og set metoder for objektvariablen

- b) Det er allerede startet å jobbe med gruppesamtaler, og i den forbindelse ble det laget tester for endepunkter samt DAO-klasse. Disse ligger i mappen `groupChat` som txt-filer og må legges på riktig sted. `GroupChatDAO` skal i dao-mappen, og testene skal i test-java mappen.
- c) Videre trenger vi endepunkter for å håndtere gruppesamtaler. Deler av `GroupChatResource.java` er skrevet, men er ikke ferdig. Dere skal nå skrive ferdig endepunktene som trengs.

Første endepunktet er for å hente ut en gruppesamtale gitt en spesifikk ID. Her er metodehode med annoteringer på plass, men selve metoden er ikke skrevet ferdig. Metoden tar inn IDen til gruppesamtalen som skal hentes som argument.

- Hent gruppesamtalen fra database ved bruk av metoden `getGroupChat(groupId)` fra `GroupChatDAO.java`.
- Meldingene som tilhører gruppesamtalen er ikke med i dette objektet og må dermed hentes ut fra databasen. Tilhørende meldinger hentes med metoden `getGroupChatMessages(groupId)`, og legges til i objektet som ble hentet ut.
- Brukerne som tilhører gruppesamtalen er ikke med i objektet som ble hentet og må dermed hentes ut fra databasen. Tilhørende brukere hentes med metoden `getGroupChatUsers(int groupId)`, og legges til i objektet som ble hentet ut.
- Returner komplett `GroupChat` objekt.

Neste endepunkt som skal lages er endepunktet for å hente gruppesamtaler som en spesifikk bruker er del av.

- Requesten er av typen GET
- Endepunkt URI skal være `user/{userId}`
- Metoden produserer JSON
- Metoden returnerer en `ArrayList` med `GroupChat` objekter

- Metoden bruker ID fra endepunktets URI og må dermed ta inn `@PathParam("userId") int userId` som argument
- Metoden bruker `getGroupChatByUserId(int userId)` for å hente gruppesamtaler fra database

Videre må vi ha et endepunkt for å lagre nye gruppesamtaler i database. Metoden skal ha navn: `postGroupChat`

- Requesten er av typen POST
- Endepunktet konsumerer JSON
- Endepunktet produserer JSON
- Metoden returnerer et objekt av typen `GroupChat`
- Metoden tar inn et objekt av typen `GroupChat` som argument
- Metoden bruker `addGroupChat(GroupChat groupChat)`, som returnerer et `GroupChat` objekt (dette skal returneres av metoden).

Vi må også ha et endepunkt for å hente ut alle meldinger tilhørende en gruppesamtale gitt ID til gruppesamtalen.

- Requesten er av typen GET
- Endepunktets URI er `{groupChatId}/message`
- Endepunktet produserer JSON
- Metoden returnerer en `ArrayList` med `Message` objekter
- Argumentet til metoden er `@PathParam("groupChatId") int groupChatId`
- Metoden bruker metoden `getGroupChatMessages(int groupChatId)` fra DAO klassen for å hente meldingene fra databasen

Til slutt trenger man et endepunkt for å lagre nye meldinger i databasen. Metoden skal ha navn: `postMessage`.

- Requesten er av typen POST
- Endepunktets URI er `{groupChatId}/message`
- Endepunktet konsumerer JSON
- Endepunktet produserer JSON
- Metoden returnerer et `Message` objekt
- Argumentet til metoden er `@PathParam("groupChatId") int groupChatId, Message message`

- Metoden bruker metoden `addMessage(int groupId, Message message)` fra DAO klassen for å lagre meldingen i databasen. Denne metoden returnerer `Message` objektet som skal sendes tilbake til klient.
- d) Det er skrevet en del javascript kode som håndterer gruppesamtaler på klient-siden. Det som mangler er et html element som gjør det mulig å lage nye gruppesamtaler. I `app.html` kan dere finne et `ul`-element som har `id="userList"`. Inne i dette elementet ligger et `li`-element som er benyttet som en knapp for å åpne kalkulatoren. Her kan dere legge til et `li`-element som håndterer å lage nye gruppesamtaler.

Legg til et `li`-element som har følgende attributt:

- `onclick` skal kalle funksjonen `openNewGroupChatForm()`
- Dere kan også legge til et ikon. Dette kan gjøres på lik måte som for elementet for kalkulatoren. For `i`-elementet kan `class` byttes til `class="fas fa-plus userIcon"`.

## Oppgave 8

En ny iterasjon er over og dere skal gjennomføre en retrospektiv. Nedenfor er det gitt punkter dere kan reflektere over. Som ved dag 1, fyll inn i malen «Retrospektiv dag 2» og send dette på mail til [surya.b.kathyat@ntnu.no](mailto:surya.b.kathyat@ntnu.no), med «Retrospektiv dag 2, gruppe nr. XX» som subject.

- Hvis vi kunne gjort iterasjonen på nytt:
  - Hva ville vi gjort annerledes?
  - Hva ville vi gjort likt?
- Hva gikk bra?
- Hva kan forbedres?
- Hvorfor er det viktig at ny funksjonalitet blir testet grundig før lansering?
- Hvorfor er det viktig med enhetstesting?
- Hvordan har samarbeidet mellom driftere og utviklere på gruppen vært?
- Har dere tips til forbedring av workshop?

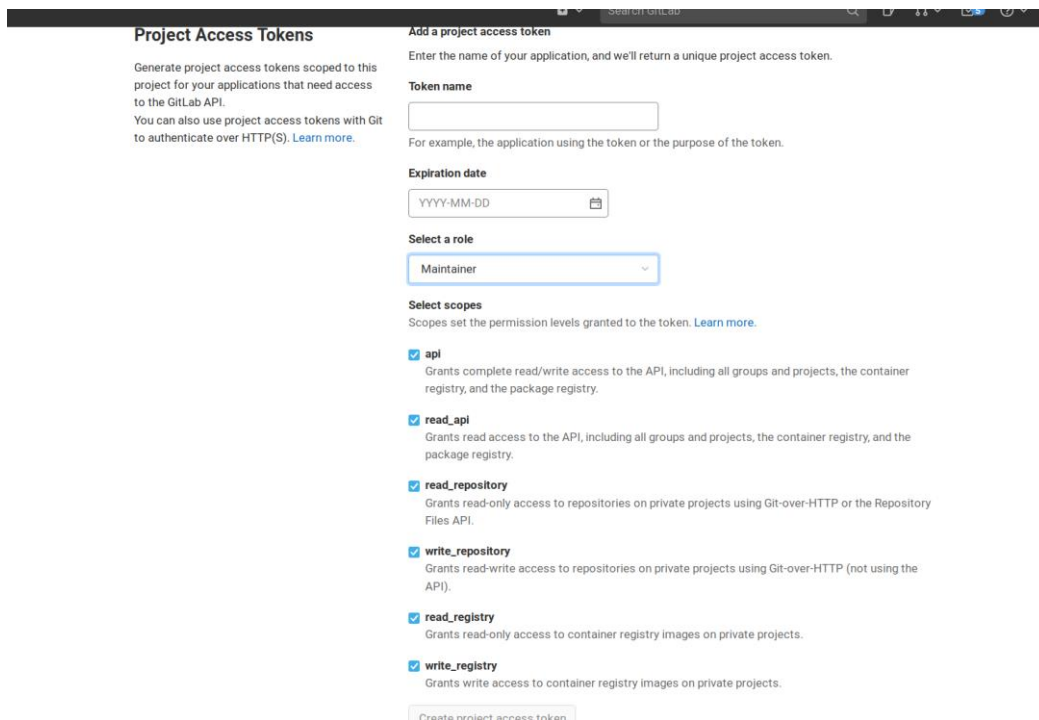


# Gitlab Access Tokens

## Opprette

**Merk:** bruk av tokens er for de som *ikke* har satt opp SSH-nøkler eller av annen grunn må benytte HTTP i forbindelse med Gitlab. I denne workshopen er vi nødt til å klonе et prosjekt ned på en VM som er delt mellom flere gruppemedlemmer, og det å bruke et access token er en naturlig kandidat, fremfor at hvert gruppemedlemmet skulle ha sine egne SSH-nøkler på VMen.

Når en benytter tofaktorautentisering - som er satt opp på Gitlab ved NTNU Gløshaugen - kan man komme bort i den situasjonen at man ikke kan benytte dette fra kommandolinja (en får kun skrevet inn brukernavn og passord, men ikke selve tofaktor-koden). Man må derfor opprette et Access Token for et gitt repo (merk: dette er pr. *repo*, ikke pr. bruker) for å kunne bruke git fra kommandolinja.



The screenshot shows the 'Add a project access token' form in GitLab. The form is titled 'Project Access Tokens' and includes instructions: 'Generate project access tokens scoped to this project for your applications that need access to the GitLab API. You can also use project access tokens with Git to authenticate over HTTP(S). [Learn more.](#)'

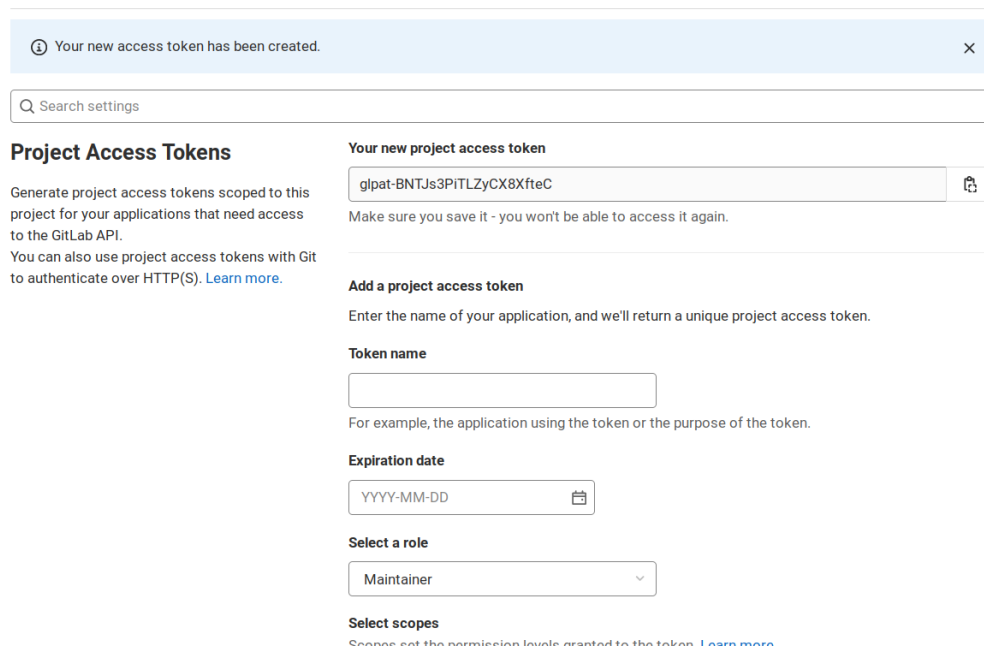
The form fields and options are:

- Token name:** A text input field with a placeholder 'Enter the name of your application, and we'll return a unique project access token.'
- Expiration date:** A date picker with a placeholder 'YYYY-MM-DD' and a calendar icon.
- Select a role:** A dropdown menu with 'Maintainer' selected.
- Select scopes:** A section with the text 'Scopes set the permission levels granted to the token. [Learn more.](#)' and a list of checkboxes:
  - ☒ **api**: Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.
  - ☒ **read\_api**: Grants read access to the API, including all groups and projects, the container registry, and the package registry.
  - ☒ **read\_repository**: Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
  - ☒ **write\_repository**: Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).
  - ☒ **read\_registry**: Grants read-only access to container registry images on private projects.
  - ☒ **write\_registry**: Grants write access to container registry images on private projects.

At the bottom of the form is a button labeled 'Create project access token'.

1. Gå til gitlab-repoet man skal opprette et access token for, og trykk på «Settings» -> «Access Tokens»
2. Fyll et navn («Token name») og evt. ekspirasjonsdato for tokenet. I tillegg må det bestemmes hvilken rolle innad i prosjektet man vil at tokenet skal ha. For workshopen så vil det fungere fint å la den stå på defaulten «Maintainer».
3. Etter dette må man huke av for «Select scopes». Vi skal ikke gå gjennom alternativene i detalj her, men for formålet i denne workshopen, kan vi simpelthen krysse av for alle.

4. Trykk på knappen «Create project access token», som vil gi et token som dette:



Project Access Tokens

Generate project access tokens scoped to this project for your applications that need access to the GitLab API. You can also use project access tokens with Git to authenticate over HTTP(S). [Learn more.](#)

Your new project access token

glpat-BNTJs3PiTLZyCX8XfteC

Make sure you save it - you won't be able to access it again.

Add a project access token

Enter the name of your application, and we'll return a unique project access token.

Token name

Expiration date

YYYY-MM-DD

Select a role

Maintainer

Select scopes

Scopes are the permission levels granted to the token. [Learn more.](#)

5. Husk at for å kunne bruke tokenet, må man ta vare på *både* navnet og selve tokenet

## Bruk fra kommandolinja

Under workshopen blir man gitt en VM som kjører headless Ubuntu. Det vil si at det eneste grensesnittet man har for å jobbe med serveren, er kommandolinja. Siden en også er nødt til å jobbe mot git under workshopen, må vi benytte access tokenet derfra.

Det kan i prinsippet gjøres på to måter:

- Direkte modifisering av fila *config* som ligger i *.git*-katalogen.
- Sendt ved navnet på tokenet og tokenet når en kloner prosjektet. Access tokenet vil da bli brukt ved de etterfølgende kommandoene som kommuniserer med serveren (*push*, f.eks.).

Alt man trenger å gjøre, er å legge token-navn og selve tokenet mellom *https* og selve adressen til repoet.

```
sander@vm:~/tmp$ git clone https://eksempelToken:glpat-BNTJs3PiTLZyCX8XfteC@gitlab.stud.idi.ntnu.no/alexholt/devops-workshop-gjgang.git
Cloning into 'devops-workshop-gjgang'...
remote: Enumerating objects: 134, done.
remote: Counting objects: 100% (134/134), done.
remote: Compressing objects: 100% (90/90), done.
remote: Total 134 (delta 40), reused 123 (delta 36), pack-reused 0
Receiving objects: 100% (134/134), 115.88 KiB | 19.31 MiB/s, done.
Resolving deltas: 100% (40/40), done.
sander@vm:~/tmp$
```

# Installere og kjøre Jenkins server

1. Koble til VM
2. Prosjektet skal allerede ha blitt klonet ned til VMen (oppgave 2), så gå i mappen “devops-workshop”
3. Sett filen ‘setupJenkins’ til å være eksekverbar

```
$ chmod +x ./setupJenkins
```

4. Kjør kommando i rotkatalogen

```
$ ./setupJenkins
```

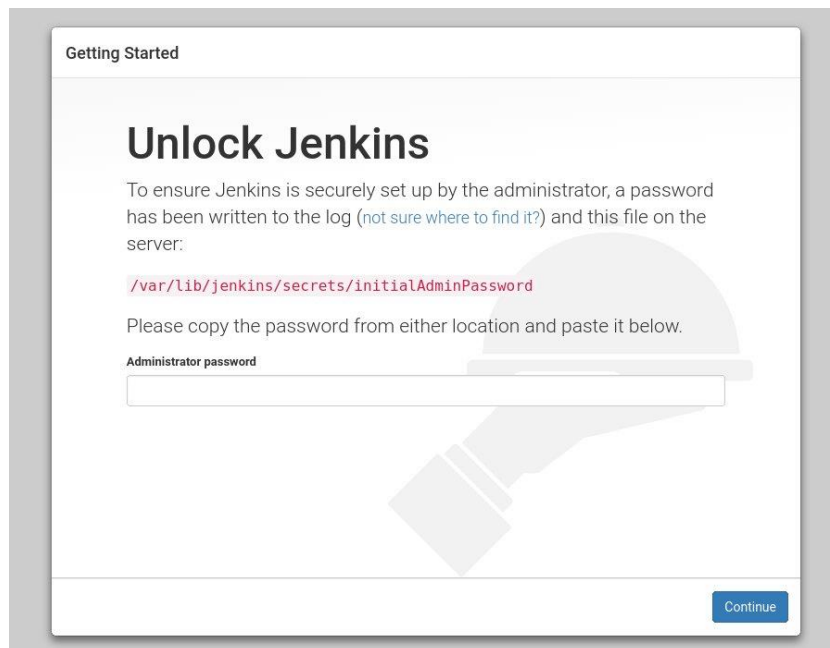
setupJenkins installerer Jenkins. Se på filen for å forstå hva som skjer. Når installasjonen er ferdig, vil passordet for å logge på Jenkins serveren stå i terminalen. Ta vare på dette.

5. Bekreft at Jenkins server er oppe og kjører ved å gå til <http://ip-til-vm:8080>

# Konfigurere Jenkins server

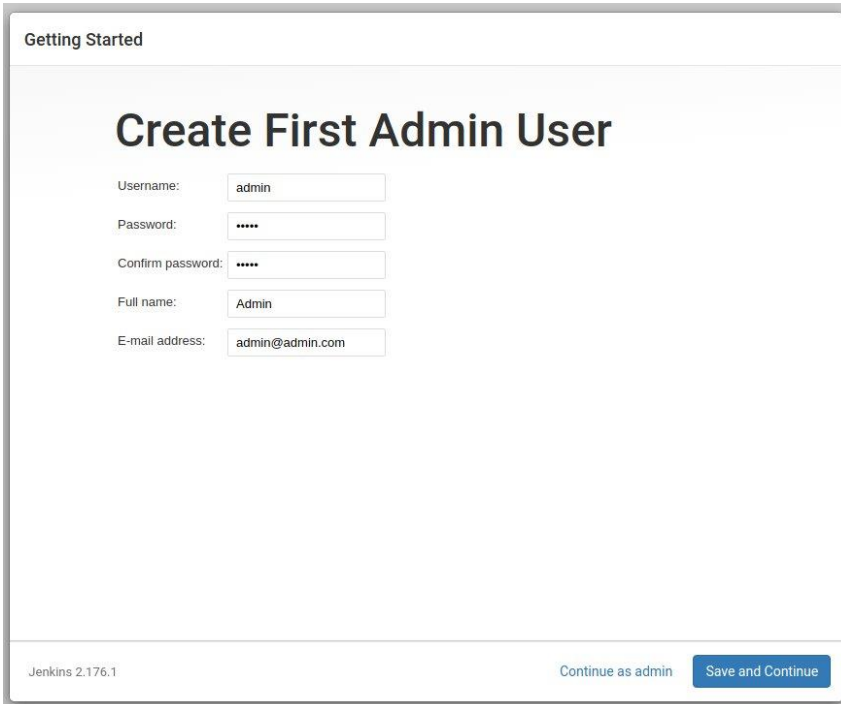
1. Fyll ut passordet som dukket opp i terminalen etter installering av Jenkins, og trykk Continue. Dersom dere trenger å hente ut passordet på nytt, kan dette gjøres med kommando:

```
$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```



2. Trykk på "Install suggested plugins", og vent til installeringen er ferdig.

3. Lag deretter en Admin bruker på Jenkins, og trykk "Save and Continue".



The screenshot shows the 'Getting Started' page in Jenkins 2.176.1. The main heading is 'Create First Admin User'. Below it are several input fields: 'Username' with 'admin', 'Password' with masked characters, 'Confirm password' with masked characters, 'Full name' with 'Admin', and 'E-mail address' with 'admin@admin.com'. At the bottom right, there are two buttons: 'Continue as admin' and 'Save and Continue'.

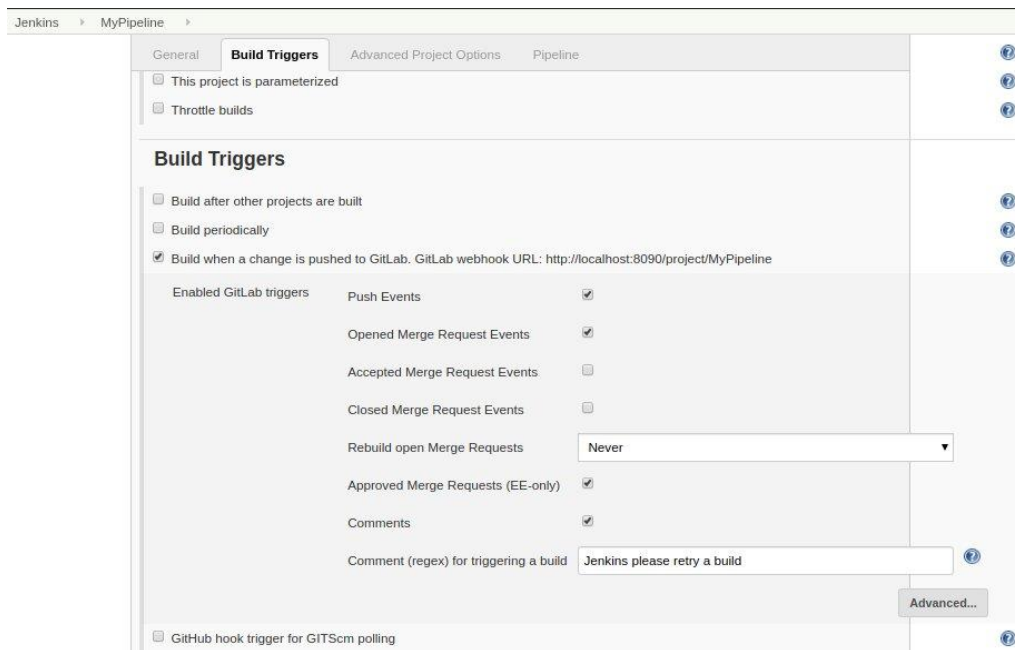
4. På neste side velges URL til Jenkins. Denne kan være slik den er. Trykk på "Save and Finish".
5. Restart Jenkins ved å gjøre følgende:
- Gå til [http://IP\\_TIL\\_VM:8080/restart](http://IP_TIL_VM:8080/restart), og velg "Yes"
  - Logg inn igjen når Jenkins server er oppe å kjører igjen (den hevder å refreshe av seg selv, men ikke stol på det)
6. For å bruke GitLab med Jenkins må vi også installere en GitLab plugin. Dette gjøres på følgende måte:
- Trykk på "Konfigurer Jenkins"/"Manage Jenkins" (på venstre side)
  - "Behandle programtillegg"/"Manage plugins"
  - Trykk på "Tilgjengelige"/"Available"
  - Søk etter GitLab
  - Huk av for GitLab
  - Søk etter Locale
  - Huk av for Locale og trykk på "Install without restart"
  - Gå til Manage Jenkins -> Configure System
  - Under «Locale», skriv «en» inn i tekstfeltet for «Default Language», og huk av for «Ignore browser preference and force this language to all users». Dette vil endre

språket til å være engelsk overalt, i stedet for å ha en hybrid av norsk og engelsk.

Trykk på Save.

7. Dere skal nå lage en pipeline hvor prosjektet deres blir automatisk compilert og rullet ut.

- Gå til forsiden og trykk på «Please create new jobs» (eller «Create new job», denne teksten kan variere litt fra utgave til utgave).
- Fyll inn et navn (f.eks. “Production”) på pipelinen i tekstetfeltet «Enter an item name».
- Trykk deretter på «Pipeline» og trykk OK.
- Under Build Triggers huk av slik som på bildet:



- Under Pipeline velg slik som på bildet under, og husk å fylle ut URL til git repository som dere forket tidligere. URLen er samme som dere bruker for å clone (HTTPS, ikke SSH).

NB! Dette er URL repoet til den på gruppa som klonet/forket repoet, IKKE

<https://gitlab.stud.idi.ntnu.no/surya/devops-workshop.git>

- Husk å bruke URL med access-token.

Jenkins > MyPipeline >

General Build Triggers Advanced Project Options **Pipeline**

**Pipeline**

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL:

Credentials: - none -

Advanced...

Add Repository

Branches to build

Branch Specifier (blank for 'any'): \*/master

Add

Branch

Repository browser: (Auto)

Additional Behaviours: Add

Script Path: Jenkinsfile

Lightweight checkout: ☒

[Pipeline Syntax](#)

Save Apply

- Scroll opp til Build Triggers igjen og kopier GitLab webhook URL. Denne trengs for å sette opp Jenkins på GitLab.
- Trykk til slutt på Save.

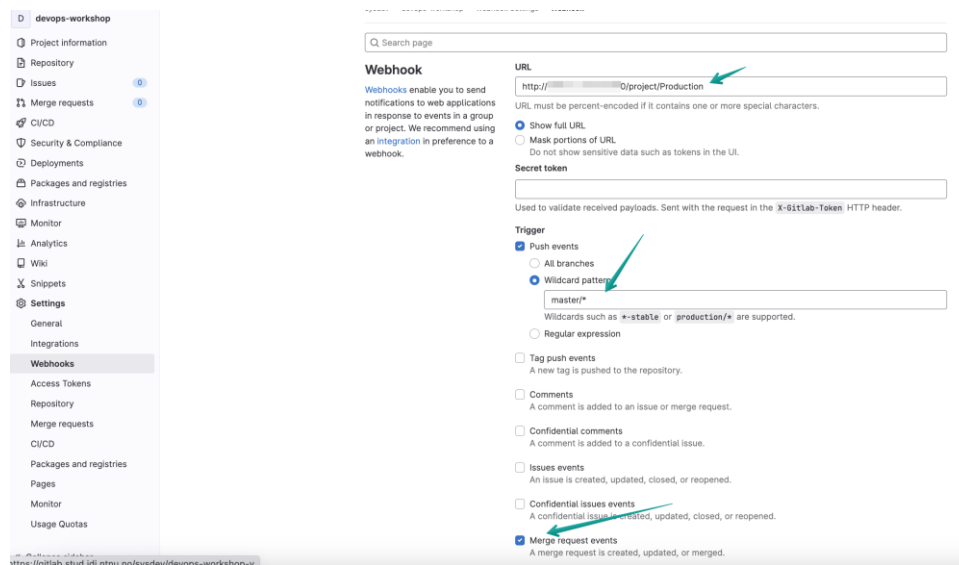
8. Gå til forsiden på Jenkins og til innstillingene:

- Manage Jenkins
- Configure System
- Scroll ned til Gitlab innstillingene
- "Enable authentication for '/project' end-point" skal ikke være huket av (checked off)
- Trykk på save

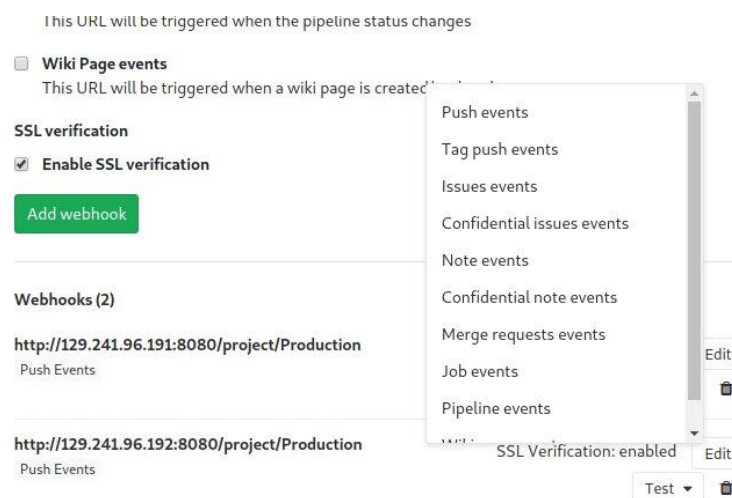
9. Konfigurer Maven på Jenkins:

- Manage Jenkins
- Global Tool Configuration
- Scroll ned til Maven og trykk på Add Maven
- Fyll ut Name: M3, og trykk enter

10. Gå inn på GitLab, Settings -> Webhooks. Etter det skal du få opp en bilde som nedenfor:



- Lim inn GitLab webhook URLen som ble kopiert fra Jenkins.
- Under Push events -> Wildcard pattern, skriv inn "master/\*".
- Velg 'Merge request events'
- Trykk på "Add webhook"
- Trykk på Test -> Push events for å teste at Webhooken fungerer som den skal:

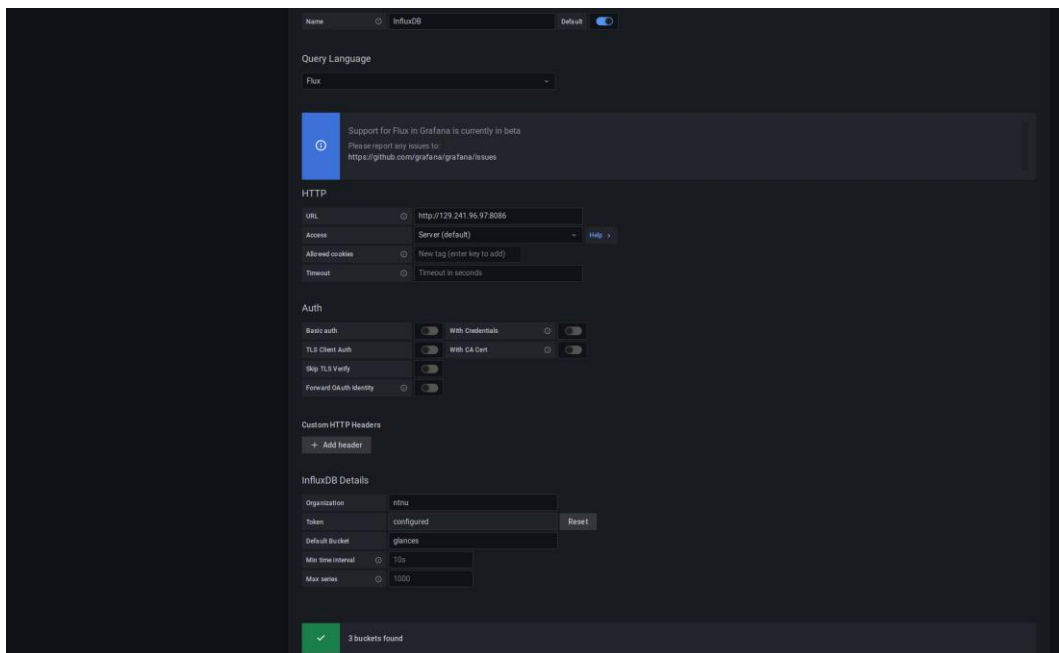


På Jenkins kan man også se at det kjøres en build. Mer om dette i oppgave 4.



# Konfigurere Grafana

1. Gå til <http://IP-TIL-VM:3000>
2. Logg inn. Brukernavn og passord er admin
3. Velg “skip”, slik at de andre på gruppa kan bruke samme innlogging senere
4. Trykk på tannhjulet og velg “Data Sources”. Velg “Add data source” og velg influxDB
5. Fyll inn feltene:
  - a. Nedtrekksmenyen «Query Language» settes til «Flux»
  - b. URL: `http://IP_TIL_VM:8086`
  - c. Knappen «Basic Auth» skal være slått av.
  - d. Organization: `ntnu`
  - e. Token: `randomTokenValue`
  - f. Default bucket: `glances`



6. Trykk på “Save & Test”. Dersom alt lykkes vil meldingen «3 buckets found» vises.
7. Under “+”-menyen velg “import” i meny på venstre side (Dashboard icon).
8. Feltet “Import via Grafana.com” skal fylles ut med 2387. Dette er et forhåndslaget dashboard for glances.
9. Trykk “load” (Dette er ikke alltid nødvendig)
10. I nedtrekksmenyen «glances» velges «InfluxDB»
11. Trykk “Import”

12. Du skal nå kunne se dashboardet:

