# Chaos Engineering

By Travis Higgins

# Overview and Explanation of Chaos Engineering

# Netflix – Pioneers of Chaos Engineering

- Netflix used to have a monolithic system architecture and it was housed in vertically scaled server racks. In 2008, this single point of weakness meant a major database corruption resulted in a 3-day downtime where DVD’s couldn’t be shipped to customers (Netflix used to ship DVD’s before their streaming service fully took off). This caused Netflix to migrate to using AWS and towards a microservice architecture. This increased complexity meant that a new approach was needed to help prevent seemingly random outages that would occur in this new system [1]. The solution: Chaos Engineering.
- The concept of chaos engineering was introduced by Netflix in 2010.
- One of the first chaos engineering tools built is called Chaos Monkey.
- **Chaos Monkey** was designed to randomly terminate instances in production to test the resilience of its systems on AWS.
- From **Chaos Monkey**, Netflix then went on to build a whole toolset of chaos engineering tools, which became known as the **Simian Army**.
- Since then, many other big tech companies such as **Amazon, Meta** and **Google** have followed suit.

# Why is Chaos Engineering Useful

- Allows companies to run experiments in production to test the fallback mechanisms in the production environment work as expected to mitigate the impact of a fault on the system and stop a potential outage.
- Test and staging environments can behave differently from production environments so running the tests in production is important.
- We will have the strongest confidence in the resilience of a system if we test the production system itself.
- Allows companies to identify faults in the production system in a controlled manner and then fix them before a real-world fault occurs which could potentially cause a complete system outage.

# Steady State – What is It?

- The way of defining the normal behaviour of the system.
- The steady state is usually a numerical metric that is monitored (such as streams per second / response time).
- There should be defined bounds for what the steady state is (e.g., a lower bound for the streams per second or a higher bound for the average response time).
- Business metrics should be monitored over technical metrics where possible for the steady state.
- For example, Netflix monitors the number of streams per second for the system’s steady state.

# What is Chaos Engineering?

- *"Chaos engineering is the process of testing a distributed computing system to ensure that it can withstand unexpected disruptions. It relies on concepts underlying chaos theory, which focus on random and unpredictable behaviour" [2].*
- Chaos engineering involves intentionally injecting faults into a distributed system running in production to test that the fallback mechanisms in place work as expected.
- Chaos Engineering takes a **black box approach** – when an experiment is run and evaluated for if the system behaves as expected, the practice is concerned with whether the system behaves as expected, not necessarily how.
- The normal and expected behaviour of the system is known as the **steady state**.
- As soon as the system falls outside of the steady state the experiment should be aborted.

# Current Examples of Chaos Engineering

- Netflix – Chaos Monkey
- Gremlin
- Chaos Mesh

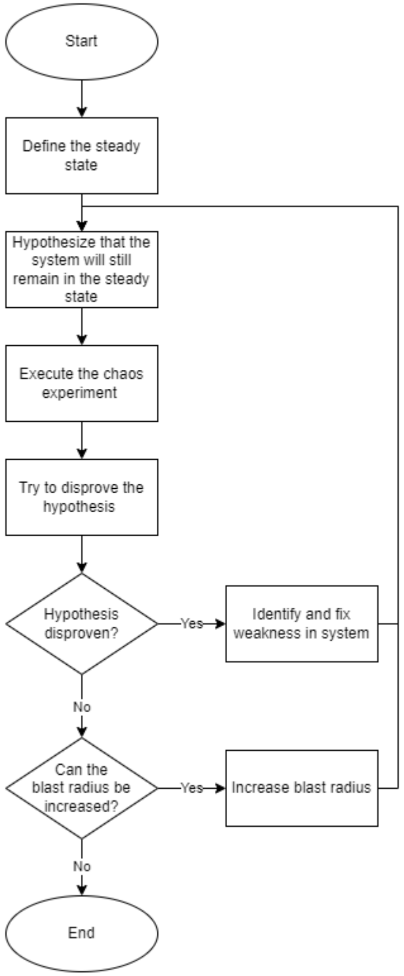
# Risks of Chaos Engineering and How to Mitigate Them

- Chaos Engineering may temporarily degrade a small subsection of some customers' experience.
- The risks of chaos engineering can be mitigated by only injecting the fault into a small portion of the system initially so that it affects fewer users.
- The size of the impact of a chaos experiment is known as the **blast radius**.
- This should initially be as small as possible and then should be incrementally increased (if the previous chaos experiment was successful) so there is a stronger confidence in the system's resilience to a fault before it is tested on a larger scale.
- This risk can also be further mitigated by initially running the chaos experiment on the testing or staging environment first before moving on to testing the production environment.

# Chaos Engineering Methodology - Background

- We have developed a chaos engineering methodology by applying and adapting current Chaos Engineering practices from both academic literature and the Principles of Chaos website [3].
- The methodology is explained on the next slide
- Following this is a walkthrough of how to set up the environment to test and how to use the chaos engineering tool.
- There is then an explanation of how to use the chaos tool to practice the methodology on the docker swarm environment that is demonstrated in the later slides

# Chaos Engineering Methodology – Explained




1. Define the steady state.
2. Hypothesise that the system will remain in its steady state (if this is not true the system should be fixed so there are fallback mechanisms in place so that we can hypothesise that the system will remain in its steady state).
3. Execute the Chaos Experiment.
4. Try to disprove the hypothesis (monitor if the system falls outside of the steady state).
5. If the hypothesis has been disproven, then identify and fix the weakness and go back to step 2.
6. If the blast radius can be increased, then increase the blast radius and go back to step 2.
7. If the blast radius cannot be increased and the hypothesis has not been disproven, then we can now have an increased confidence in the system's resilience to the fault that was tested.

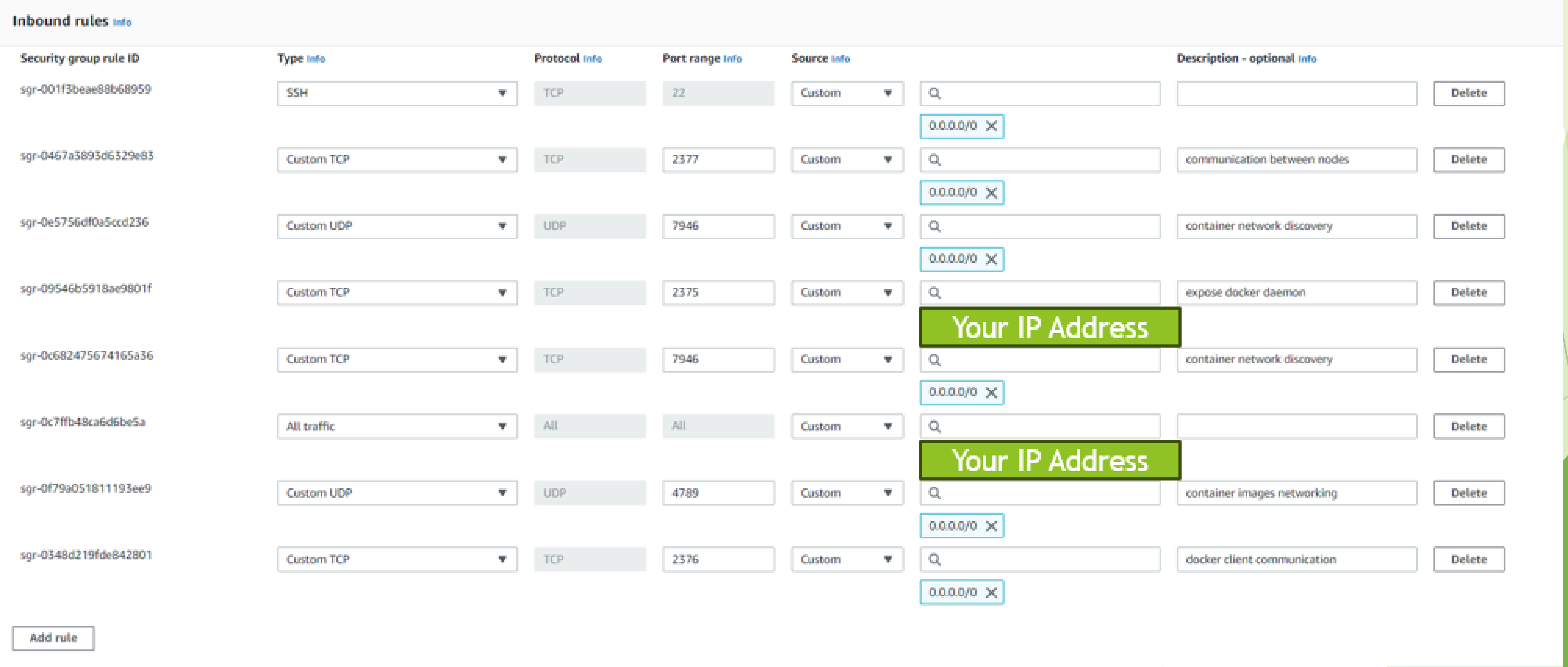
# How to Use a Chaos Engineering Tool to Practice Chaos Engineering on a Docker Swarm

# Setting up the Environment to Test – Overview

1. Download the Docker Swarm Chaos Engineering found here (https://github.com/Travis200/Chaos_Engineering_Tool_Internship)
2. Create an AWS account if you do not already have one and sign in (https://aws.amazon.com)
3. Create a security group as shown on the next slide (Modify to your own IP Address).
4. Create AWS EC2 instances.
5. Install Docker on each EC2.
6. Modify so the docker daemon is exposed.
7. Restart docker daemon.
8. Copy across locustfile.py, docker-compose.yml and prometheus.yml to the master node.
9. Pull chaos engineering image.
10. Add grafana=True label to manager node of docker swarm.
11. Deploy the stack to swarm.

# Setting up the Environment to Test – Security Group

1. Download the Docker Swarm Chaos Engineering found here (https://github.com/Travis200/Chaos_Engineering_Tool_Internship)
2. Create an AWS account if you do not already have one and sign in (https://aws.amazon.com)
3. Create a security group as shown (Modify two of the options to your IP Address, with the source dropdown)



**Inbound rules** Info

| Security group rule ID | Type Info | Protocol Info | Port range Info | Source Info | | Description - optional Info | |
|---|---|---|---|---|---|---|---|
| sgr-001f3beae88b68959 | SSH | TCP | 22 | Custom | 0.0.0.0/0 | | Delete |
| sgr-0467a3893d6329e83 | Custom TCP | TCP | 2377 | Custom | 0.0.0.0/0 | communication between nodes | Delete |
| sgr-0e5756df0a5ccd236 | Custom UDP | UDP | 7946 | Custom | 0.0.0.0/0 | container network discovery | Delete |
| sgr-09546b5918ae9801f | Custom TCP | TCP | 2375 | Custom | Your IP Address | expose docker daemon | Delete |
| sgr-0c682475674165a36 | Custom TCP | TCP | 7946 | Custom | 0.0.0.0/0 | container network discovery | Delete |
| sgr-0c7ffb48ca6d6be5a | All traffic | All | All | Custom | Your IP Address | | Delete |
| sgr-0f79a051811193ee9 | Custom UDP | UDP | 4789 | Custom | 0.0.0.0/0 | container images networking | Delete |
| sgr-0348d219fde842801 | Custom TCP | TCP | 2376 | Custom | 0.0.0.0/0 | docker client communication | Delete |

Add rule

# Setting up the Environment to Test – Create AWS Instances

1. Create AWS EC2, by selecting the launch instances.

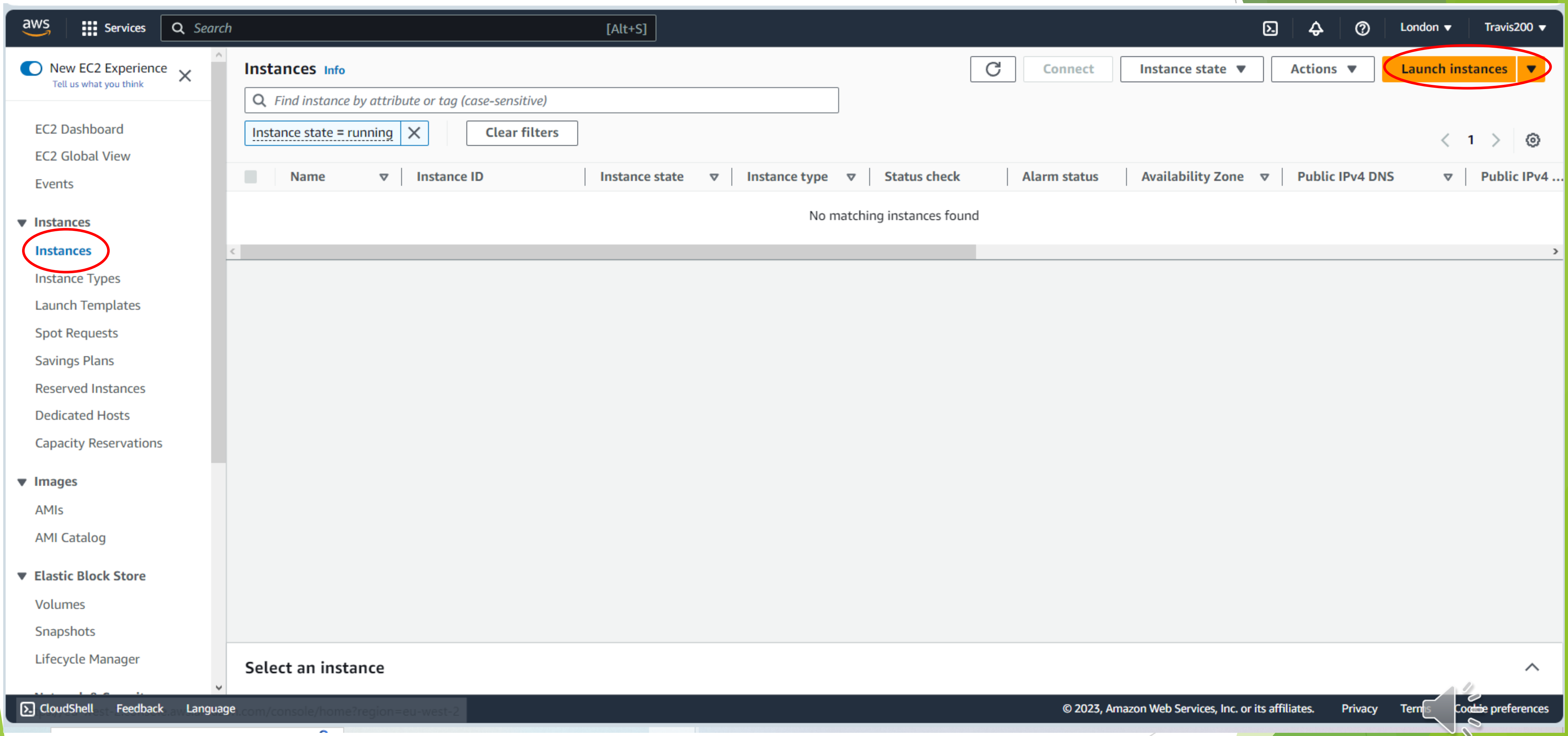
2. Name the instances.

3. Set the number of instances to 4.

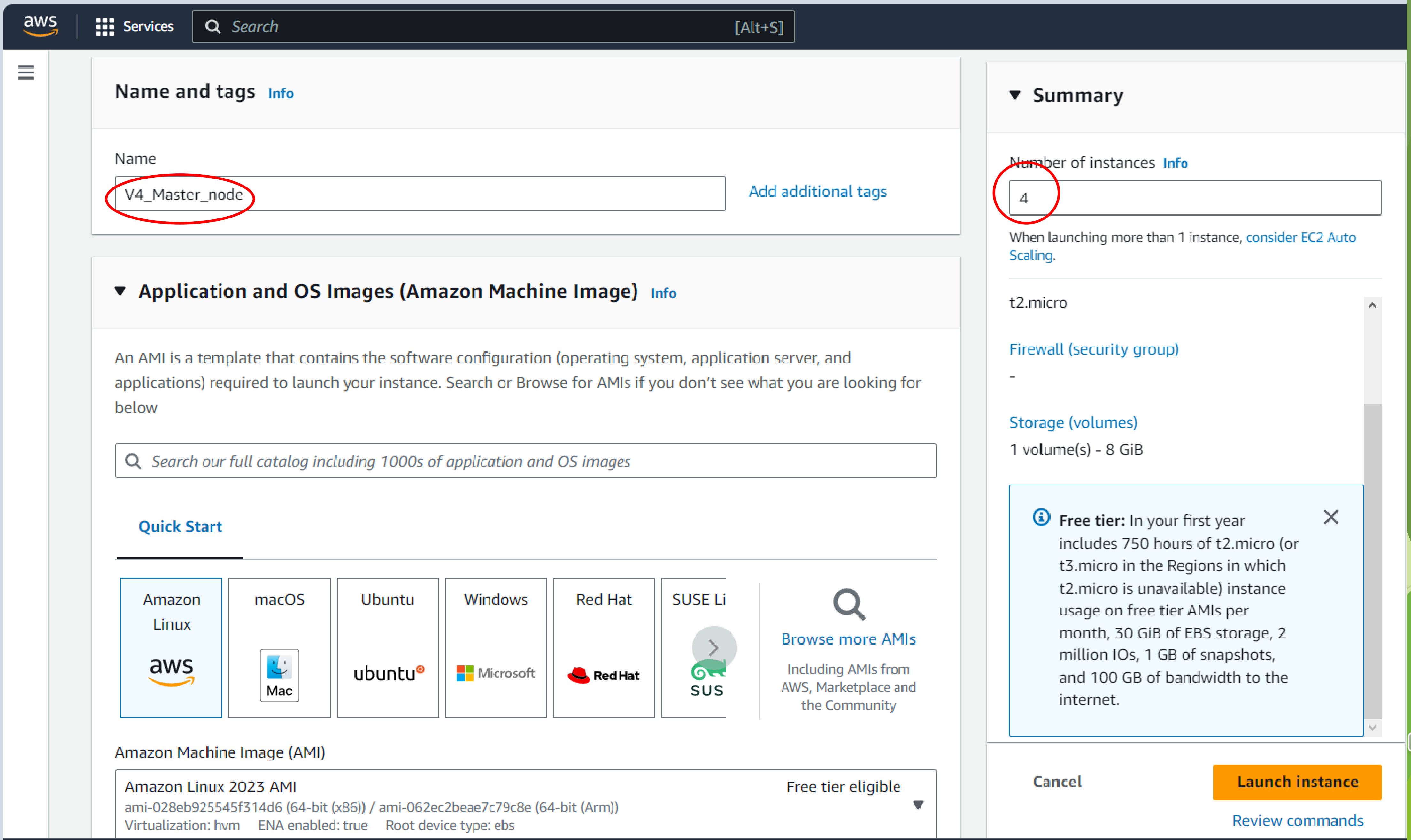
4. Create a key pair with RSA encryption and use .pem private key file format (save this in a secure location in case you wish to use Putty or WinSCP).

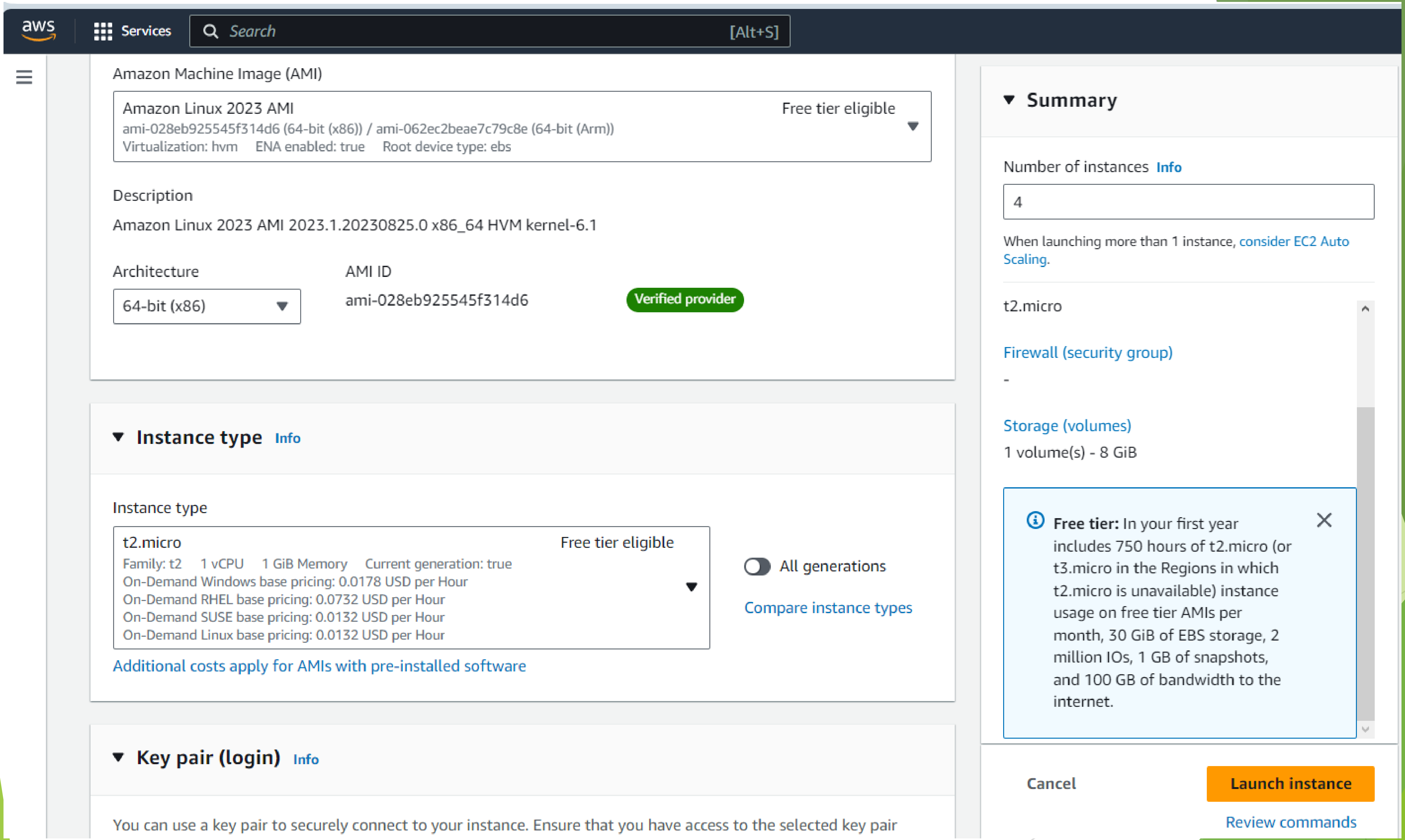
5. Select the security group you created earlier.

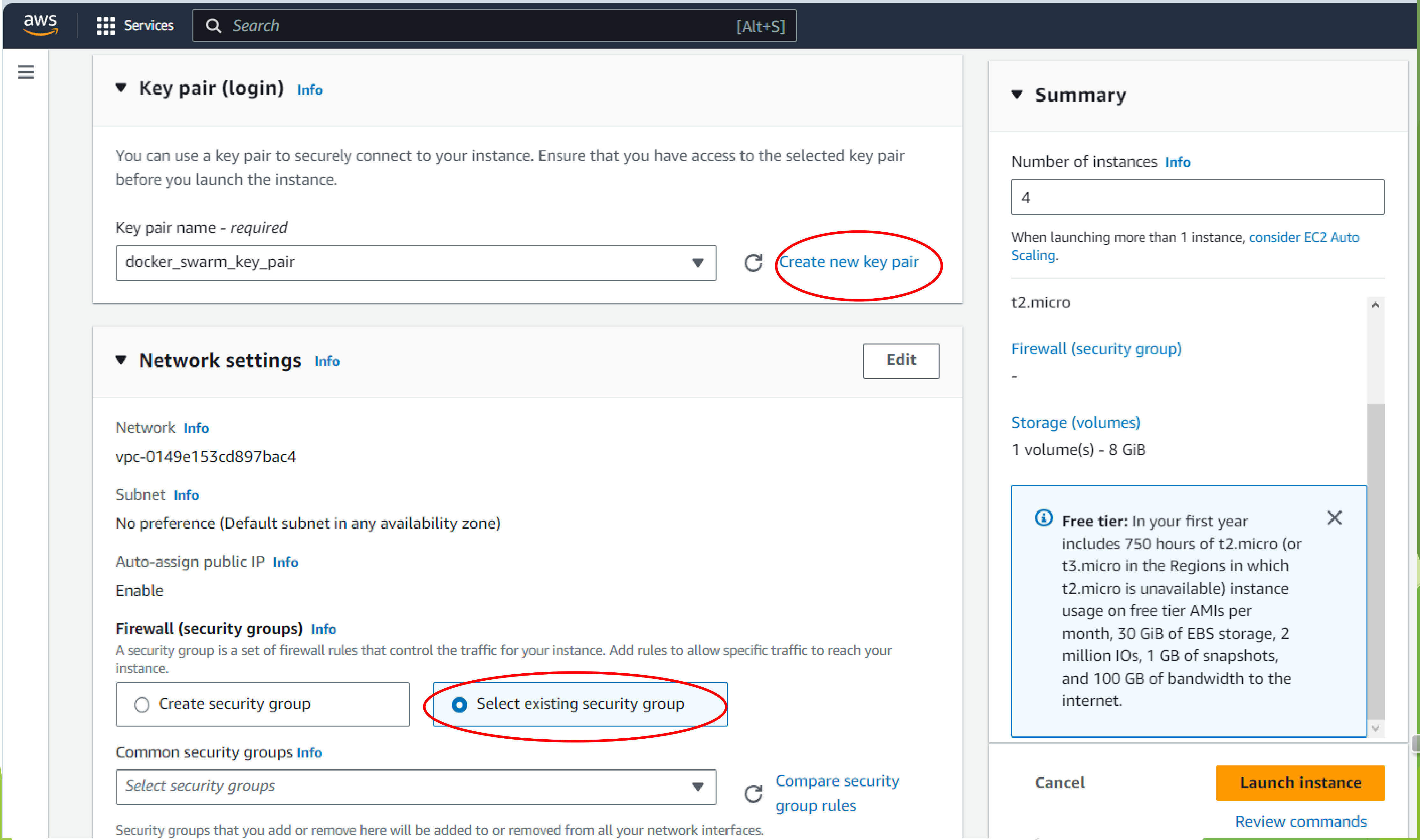
6. Rename the instances to master node and worker node 1, worker node 2, worker node 3 etc (or something similar) on the instances tab once they are created.

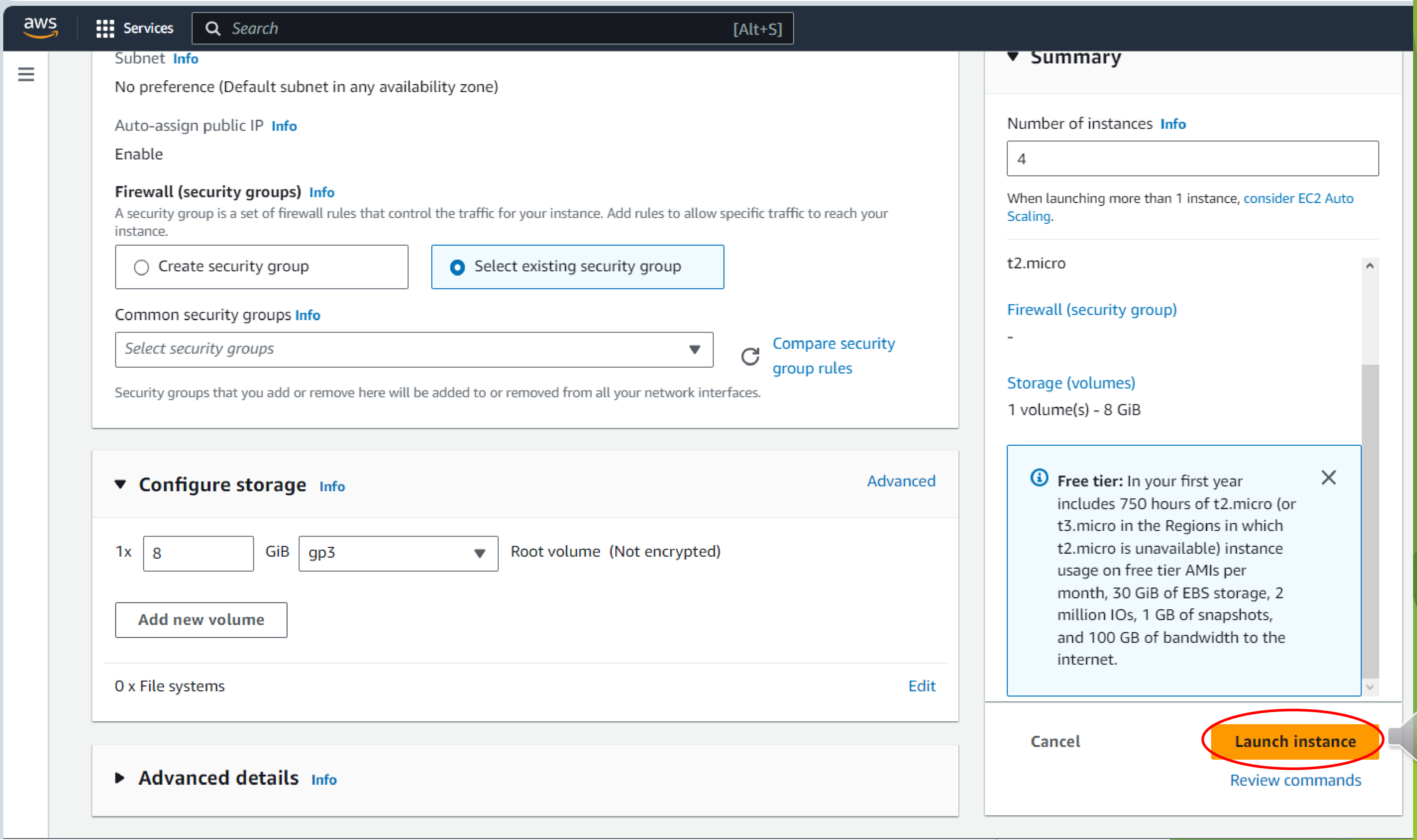


aws
Services
Search
[Alt+S]
London
Travis200
New EC2 Experience
Tell us what you think
EC2 Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Instances Info
Connect
Instance state
Actions
Launch instances
Find instance by attribute or tag (case-sensitive)
Instance state = running
Clear filters
1
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
No matching instances found
Select an instance
CloudShell
Feedback
Language
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy


aws
Services
Search
[Alt+S]
Name and tags Info
Name
V4_Master_node
Add additional tags
Application and OS Images (Amazon Machine Image) Info
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below
Search our full catalog including 1000s of application and OS images
Quick Start
Amazon Linux
aws
macOS
Mac
Ubuntu
ubuntu
Windows
Microsoft
Red Hat
Red Hat
SUSE Li
SUS
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
Amazon Linux 2023 AMI
Free tier eligible
ami-028eb925545f314d6 (64-bit (x86)) / ami-062ec2beae7c79c8e (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs
Summary
Number of instances Info
4
When launching more than 1 instance, consider EC2 Auto Scaling.
t2.micro
Firewall (security group)
-
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.
Cancel
Launch instance
Review commands

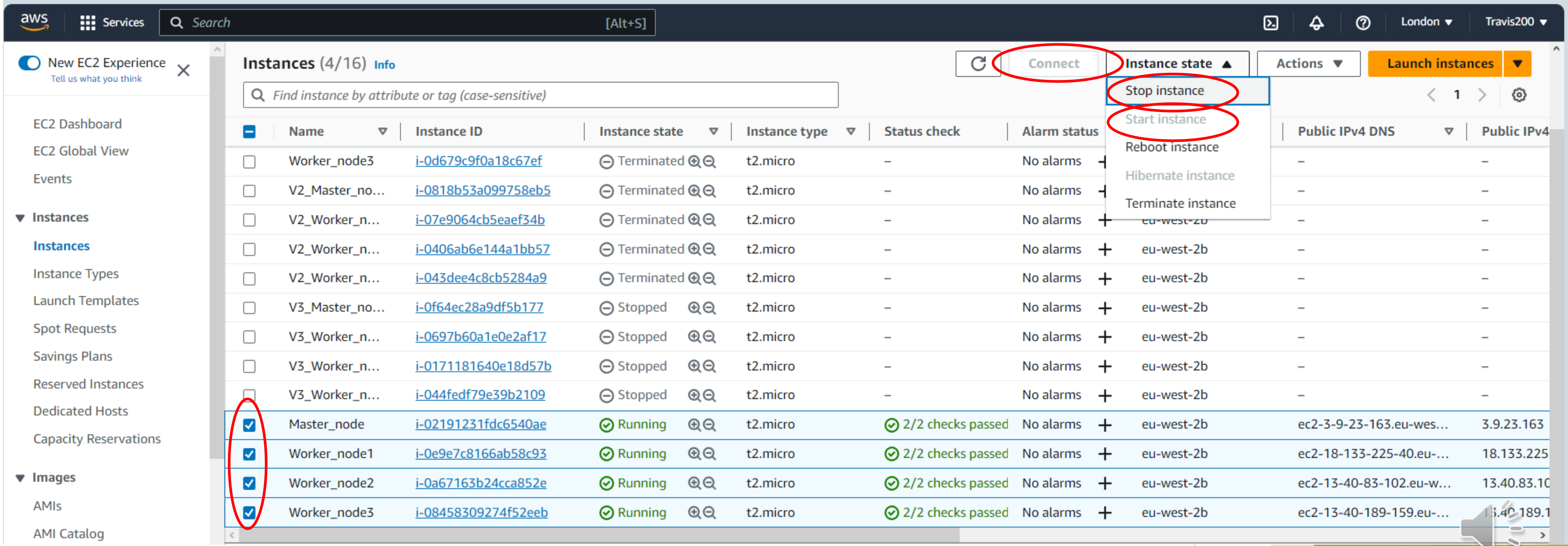

aws
Services
Search
[Alt+S]
Amazon Machine Image (AMI)
Amazon Linux 2023 AMI
Free tier eligible
ami-028eb925545f314d6 (64-bit (x86)) / ami-062ec2beae7c79c8e (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs
Description
Amazon Linux 2023 AMI 2023.1.20230825.0 x86_64 HVM kernel-6.1
Architecture
64-bit (x86)
AMI ID
ami-028eb925545f314d6
Verified provider
Instance type Info
Instance type
t2.micro
Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Windows base pricing: 0.0178 USD per Hour
On-Demand RHEL base pricing: 0.0732 USD per Hour
On-Demand SUSE base pricing: 0.0132 USD per Hour
On-Demand Linux base pricing: 0.0132 USD per Hour
All generations
Compare instance types
Additional costs apply for AMIs with pre-installed software
Key pair (login) Info
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair
Summary
Number of instances Info
4
When launching more than 1 instance, consider EC2 Auto Scaling.
t2.micro
Firewall (security group)
-
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.
Cancel
Launch instance
Review commands


aws
Services
Search
[Alt+S]
Key pair (login) Info
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
Key pair name - required
docker_swarm_key_pair
Create new key pair
Network settings Info
Edit
Network Info
vpc-0149e153cd897bac4
Subnet Info
No preference (Default subnet in any availability zone)
Auto-assign public IP Info
Enable
Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group
Select existing security group
Common security groups Info
Select security groups
Compare security group rules
Security groups that you add or remove here will be added to or removed from all your network interfaces.
Summary
Number of instances Info
4
When launching more than 1 instance, consider EC2 Auto Scaling.
t2.micro
Firewall (security group)
-
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.
Cancel
Launch instance
Review commands


aws
Services
Search
[Alt+S]
Subnet Info
No preference (Default subnet in any availability zone)
Auto-assign public IP Info
Enable
Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group
Select existing security group
Common security groups Info
Select security groups
Compare security group rules
Security groups that you add or remove here will be added to or removed from all your network interfaces.
Configure storage Info
Advanced
1x
8
GiB
gp3
Root volume (Not encrypted)
Add new volume
0 x File systems
Edit
Advanced details Info
Summary
Number of instances Info
4
When launching more than 1 instance, consider EC2 Auto Scaling.
t2.micro
Firewall (security group)
-
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.
Cancel
Launch instance
Review commands

# Setting up the Environment to Test – Connecting to AWS instance terminal

1. If the created instances aren't running, then select all of the created instances and then start all of the instances from the instance state dropdown menu.
2. Connect to each instance either via Putty or through the AWS connect button – AWS connect is recommended as the easiest method.


aws
Services
Search
[Alt+S]
London
Travis200
New EC2 Experience
Tell us what you think
EC2 Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Instances (4/16) Info
Connect
Instance state
Actions
Launch instances
Find instance by attribute or tag (case-sensitive)
Stop instance
Start instance
Reboot instance
Hibernate instance
Terminate instance
1
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Public IPv4 DNS
Public IPv4
Worker_node3 i-0d679c9f0a18c67ef Terminated t2.micro – No alarms – –
V2_Master_no… i-0818b53a099758eb5 Terminated t2.micro – No alarms – –
V2_Worker_n… i-07e9064cb5eaef34b Terminated t2.micro – No alarms – –
V2_Worker_n… i-0406ab6e144a1bb57 Terminated t2.micro – No alarms eu-west-2b – –
V2_Worker_n… i-043dee4c8cb5284a9 Terminated t2.micro – No alarms eu-west-2b – –
V3_Master_no… i-0f64ec28a9df5b177 Stopped t2.micro – No alarms eu-west-2b – –
V3_Worker_n… i-0697b60a1e0e2af17 Stopped t2.micro – No alarms eu-west-2b – –
V3_Worker_n… i-0171181640e18d57b Stopped t2.micro – No alarms eu-west-2b – –
V3_Worker_n… i-044fedf79e39b2109 Stopped t2.micro – No alarms eu-west-2b – –
Master_node i-02191231fdc6540ae Running t2.micro 2/2 checks passed No alarms eu-west-2b ec2-3-9-23-163.eu-wes… 3.9.23.163
Worker_node1 i-0e9e7c8166ab58c93 Running t2.micro 2/2 checks passed No alarms eu-west-2b ec2-18-133-225-40.eu-… 18.133.225
Worker_node2 i-0a67163b24cca852e Running t2.micro 2/2 checks passed No alarms eu-west-2b ec2-13-40-83-102.eu-w… 13.40.83.10
Worker_node3 i-08458309274f52eeb Running t2.micro 2/2 checks passed No alarms eu-west-2b ec2-13-40-189-159.eu-…

# Setting up the Environment to Test – Install Docker on each Instance

- In the terminal for each instance run the following 4 commands to install docker:

1. *sudo yum update*
2. *sudo yum install docker*
3. *sudo service docker start*
4. *sudo usermod -a -G docker ec2-user*
5. Restart the instance

# Setting up the Environment to Test – Exposing the Docker Daemon

We need to expose the docker daemon on the master node which the chaos engineering tool will be connecting to. To do this execute the following steps:

1. *sudo nano /lib/systemd/system/docker.service*
2. Look for ExecStart and change as follows:

   From: *ExecStart=/usr/bin/dockerd -H fd:// -- containerd=/run/containerd/containerd.sock*

   To: *ExecStart=/usr/bin/dockerd -H fd:// -H=tcp://0.0.0.0:2375*
3. Save and exit (ctrl + s then ctrl + x) the file and then run the following commands to restart the docker daemon:

   1. *sudo systemctl daemon-reload*

   2. *sudo service docker restart*

```
  GNU nano 5.8                                         /lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target docker.socket firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket

[Service]
Type=notify
EnvironmentFile=-/etc/sysconfig/docker
EnvironmentFile=-/etc/sysconfig/docker-storage
EnvironmentFile=-/run/docker/runtimes.env
ExecStartPre=/bin/mkdir -p /run/docker
ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock $OPTIONS $DOCKER_STORAGE_OPTIONS $DOCKER_ADD_RUNTIMES
ExecReload=/bin/kill -s HUP $MAINPID
LimitNOFILE=infinity
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNPROC=infinity
LimitCORE=infinity
# Uncomment TasksMax if your systemd version supports it.
# Only systemd 226 and above support this version.

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark   M-] To Bracket M-Q Previous   ^B Back
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy       ^Q Where Was   M-W Next       ^F Forward
```

# Setting up the Environment to Test – Copy across relevant files for chaos experiments

- Clone chaos engineering tool from the Git repository
- The repo contains both the chaos engineering tool and the docker-compose.yml file and prometheus.yml file for the Java Benchmark App (from the CSC8110 coursework)
- Copy the docker-compose.yml and prometheus.yml file across to the master node.
- This can be done through WinSCP or by the following commands:
- *touch docker-compose.yml Prometheus.yml*
- *nano docker-compose.yml* (then paste code here and then ctrl+s then ctrl + x)
- *nano prometheus.yml* (then paste code here and then ctrl+s then ctrl+x)
- On each AWS instance run the command *docker pull travis1220/chaos-engineering*

# Setting up the Environment to Test – Initialise the Docker Swarm

1. On the master node run the command *docker swarm init*
2. Copy the *docker swarm join …* and paste this into the terminal on the worker nodes
3. On the master node run the command *docker node ls* to identify the master node
4. Then run the command *docker node update --label-add grafana=True [master node ID]*
5. On the master node run the command *docker stack deploy –c docker-compose.yml dockerswarm* to deploy the stack
6. After running this command, you can use *docker service ls* to see the running services

aws Services Search [Alt+S] London ▾ Travis200 ▾

```
[ec2-user@ip-172-31-41-147 ~]$ docker swarm init
Swarm initialized: current node (neca5bu895z7po6qwmyyedbvm) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-5fotkz8q5gupltifh15bhfm1n811itvtdr0bx2edop9tb9b1r2-04enjue6xrt72nkhazcogfdqi 172.31.41.147:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

[ec2-user@ip-172-31-41-147 ~]$
```

Paste this into the other AWS instances to add them as worker nodes to the swarm

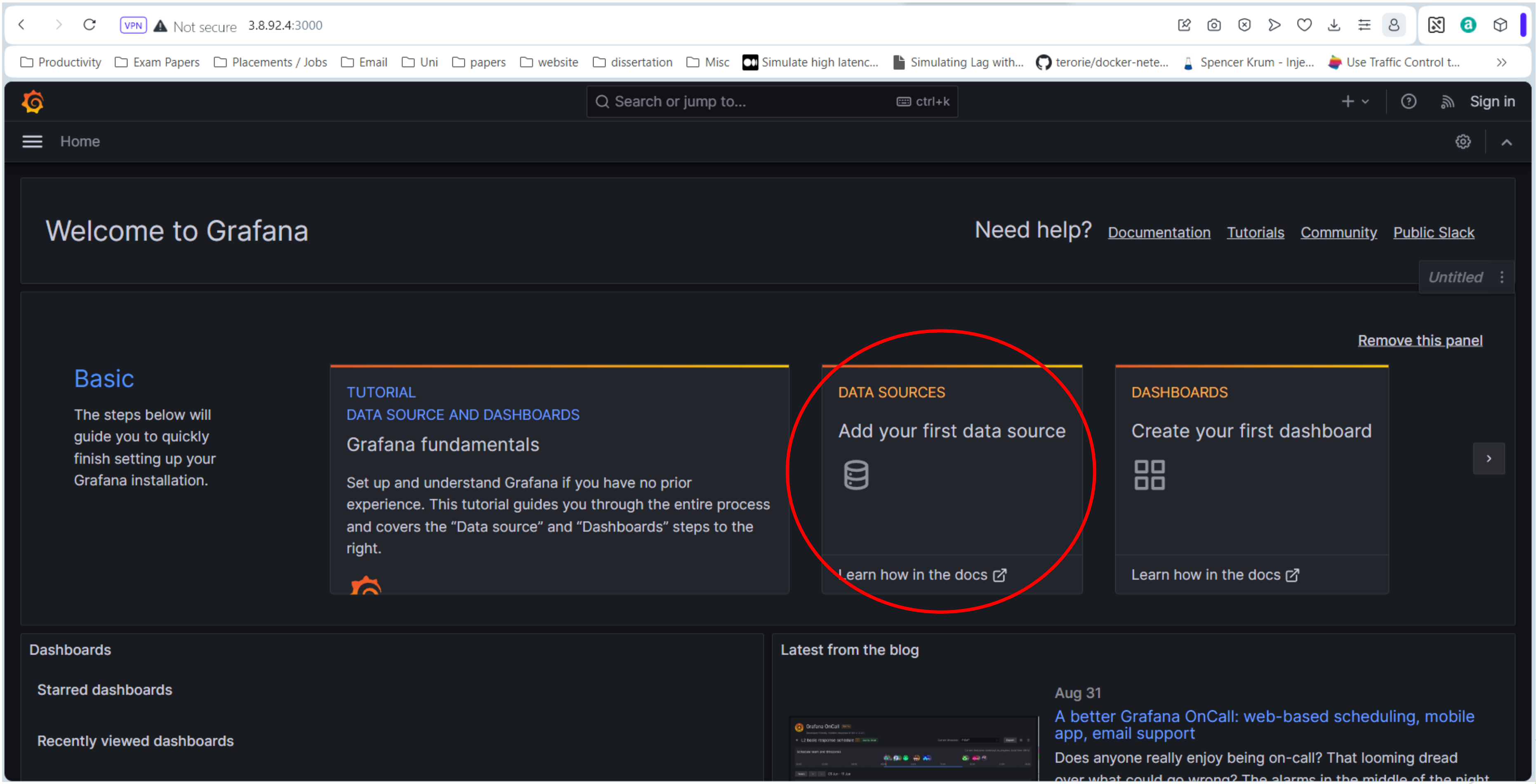
```
[ec2-user@ip-172-31-41-147 ~]$ docker node ls
ID                            HOSTNAME                                       STATUS    AVAILABILITY   MANAGER STATUS
z8p5mw4ag5vg6qaf391h8u10g     ip-172-31-33-217.eu-west-2.compute.internal    Ready     Active
pmxndr31dpcnu5xfjfh4kh0zz     ip-172-31-37-232.eu-west-2.compute.internal    Ready     Active
zpwq76904mys2wy04rfedf19u *   ip-172-31-41-147.eu-west-2.compute.internal    Ready     Active         Leader
swbhiwtuudyli29erhtx4ns3i     ip-172-31-43-22.eu-west-2.compute.internal     Ready     Active
[ec2-user@ip-172-31-41-147 ~]$ docker node update --label-add grafana=True zpwq76904mys2wy04rfedf19u
```

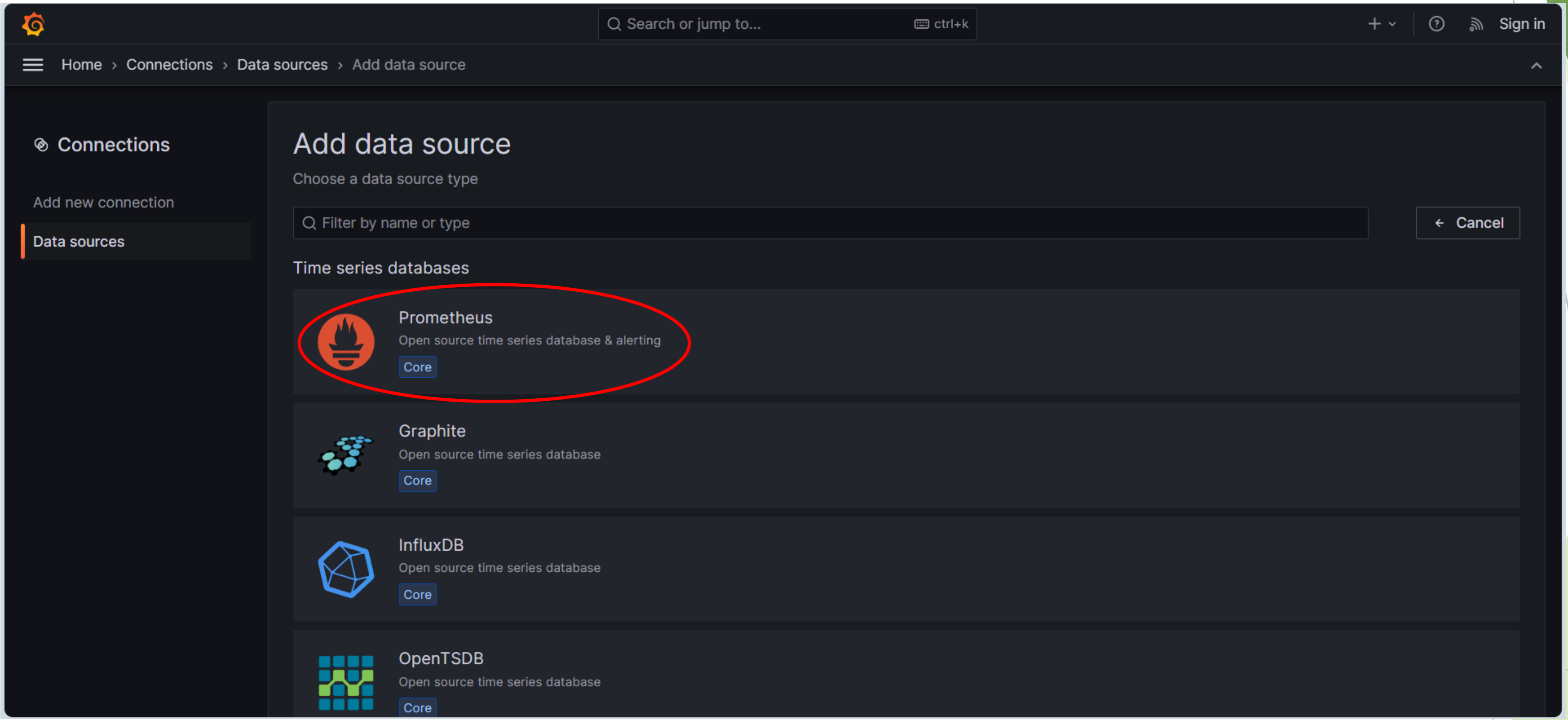
# How to Set Up Monitoring – Opening Grafana

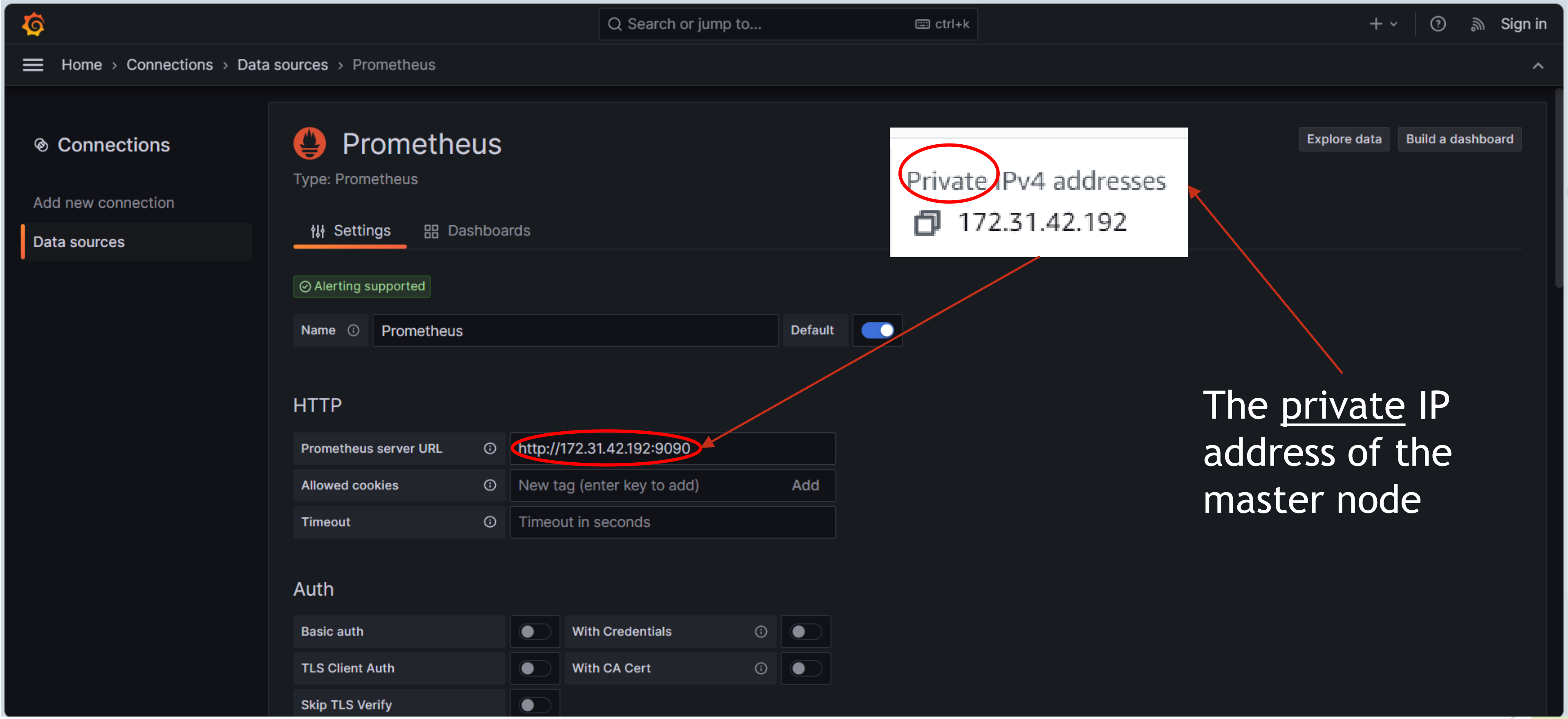
- The monitoring is integrated into the application that is tested
- If the docker swarm is running Grafana can be easily accessed in a web browser at *http://[master node IP]:3000*

# How to Set Up Monitoring – Configuring Grafana

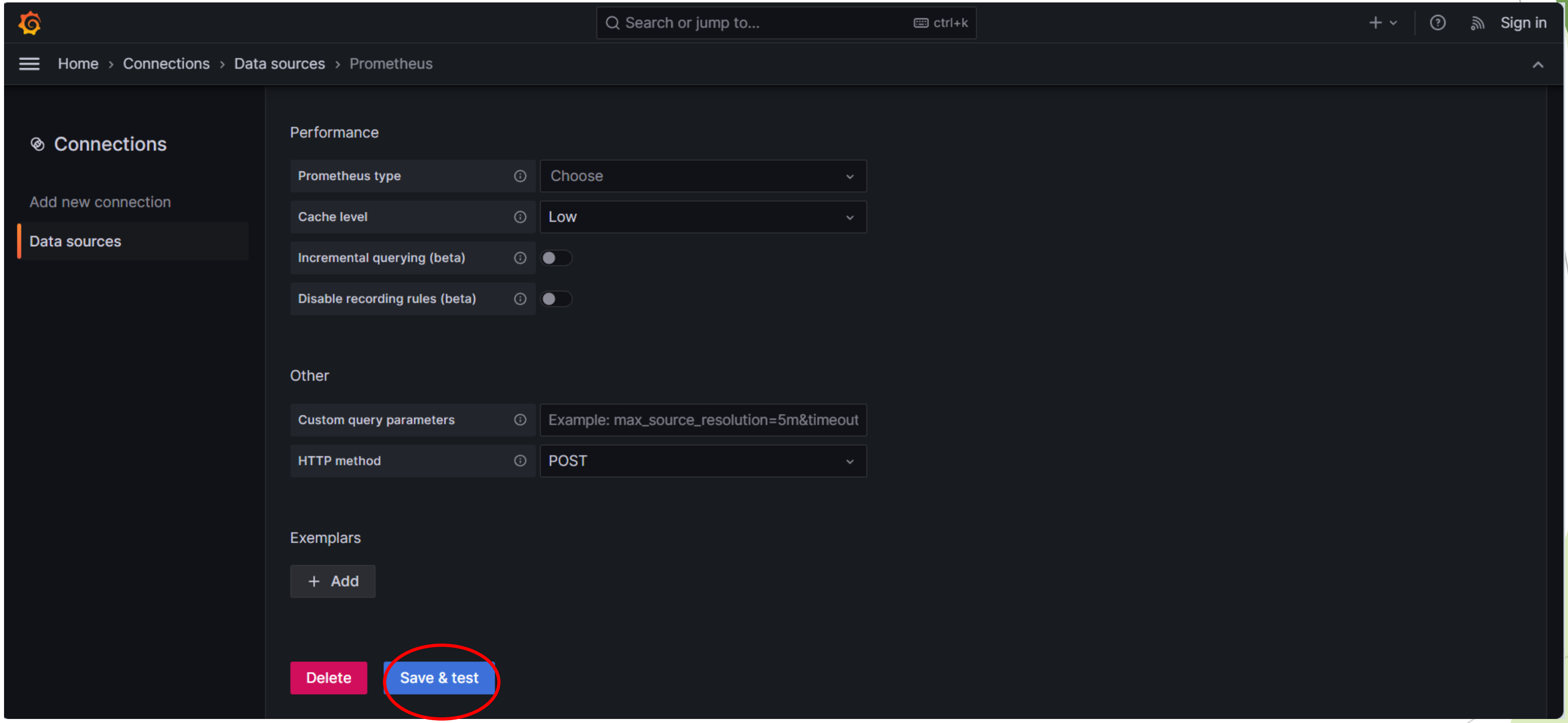
- Follow the visual instructions on the following slides

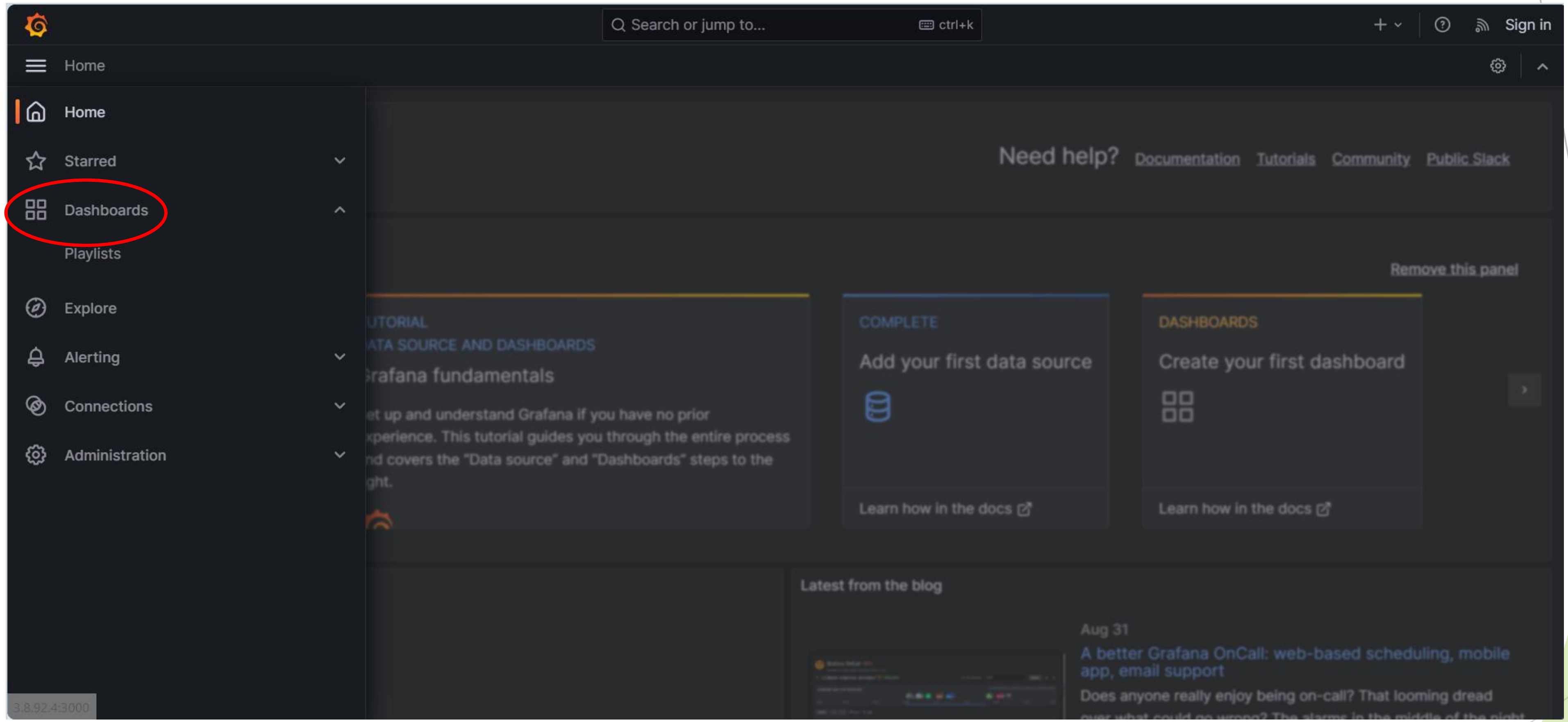

Not secure 3.8.92.4:3000
Productivity
Exam Papers
Placements / Jobs
Email
Uni
papers
website
dissertation
Misc
Simulate high latenc...
Simulating Lag with...
terorie/docker-nete...
Spencer Krum - Inje...
Use Traffic Control t...
Search or jump to...
ctrl+k
Sign in
Home
Welcome to Grafana
Need help?
Documentation
Tutorials
Community
Public Slack
Untitled
Remove this panel
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals
Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.
DATA SOURCES
Add your first data source
Learn how in the docs
DASHBOARDS
Create your first dashboard
Learn how in the docs
Dashboards
Starred dashboards
Recently viewed dashboards
Latest from the blog
Aug 31
A better Grafana OnCall: web-based scheduling, mobile app, email support
Does anyone really enjoy being on-call? That looming dread


Search or jump to...
ctrl+k
Sign in
Home › Connections › Data sources › Add data source
Connections
Add new connection
Data sources
Add data source
Choose a data source type
Filter by name or type
Cancel
Time series databases
Prometheus
Open source time series database & alerting
Core
Graphite
Open source time series database
Core
InfluxDB
Open source time series database
Core
OpenTSDB
Open source time series database
Core

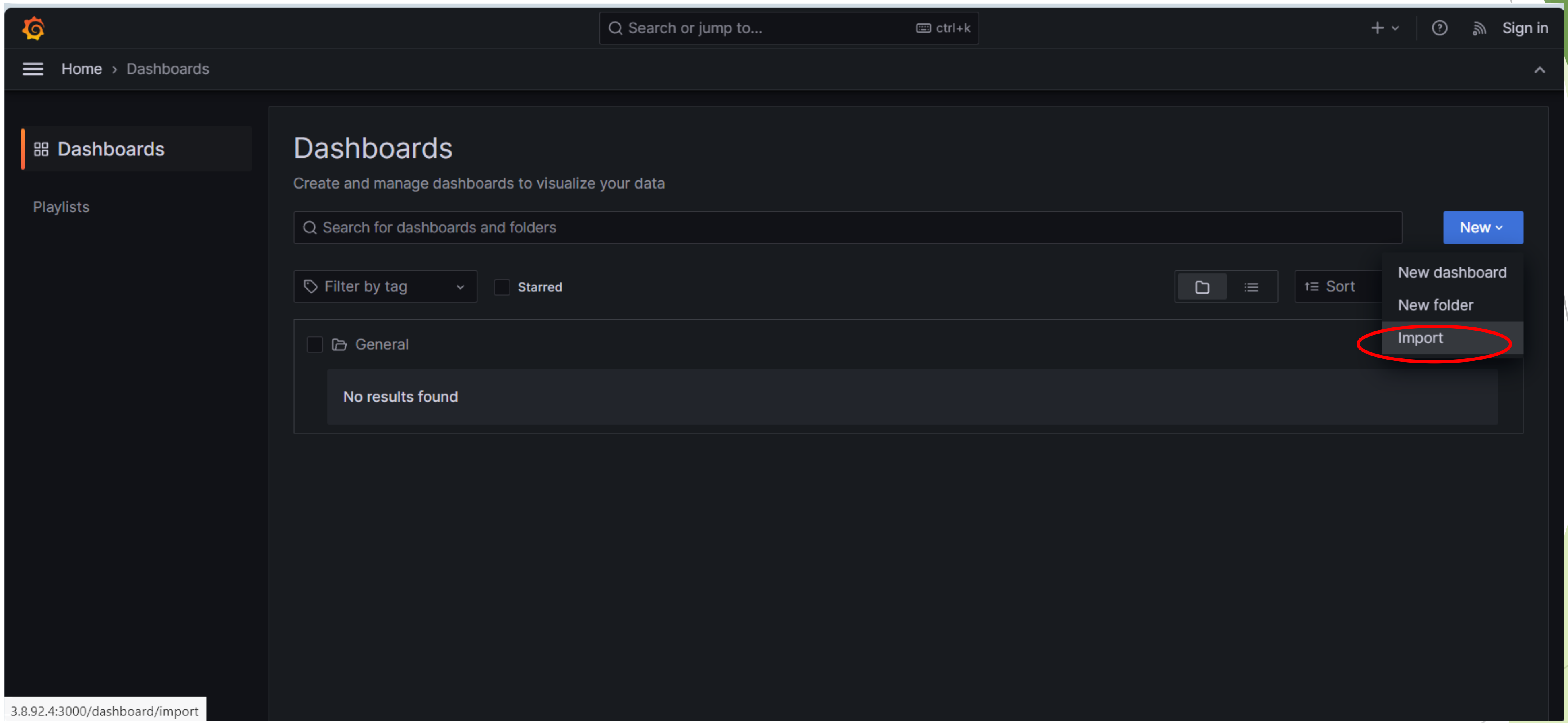


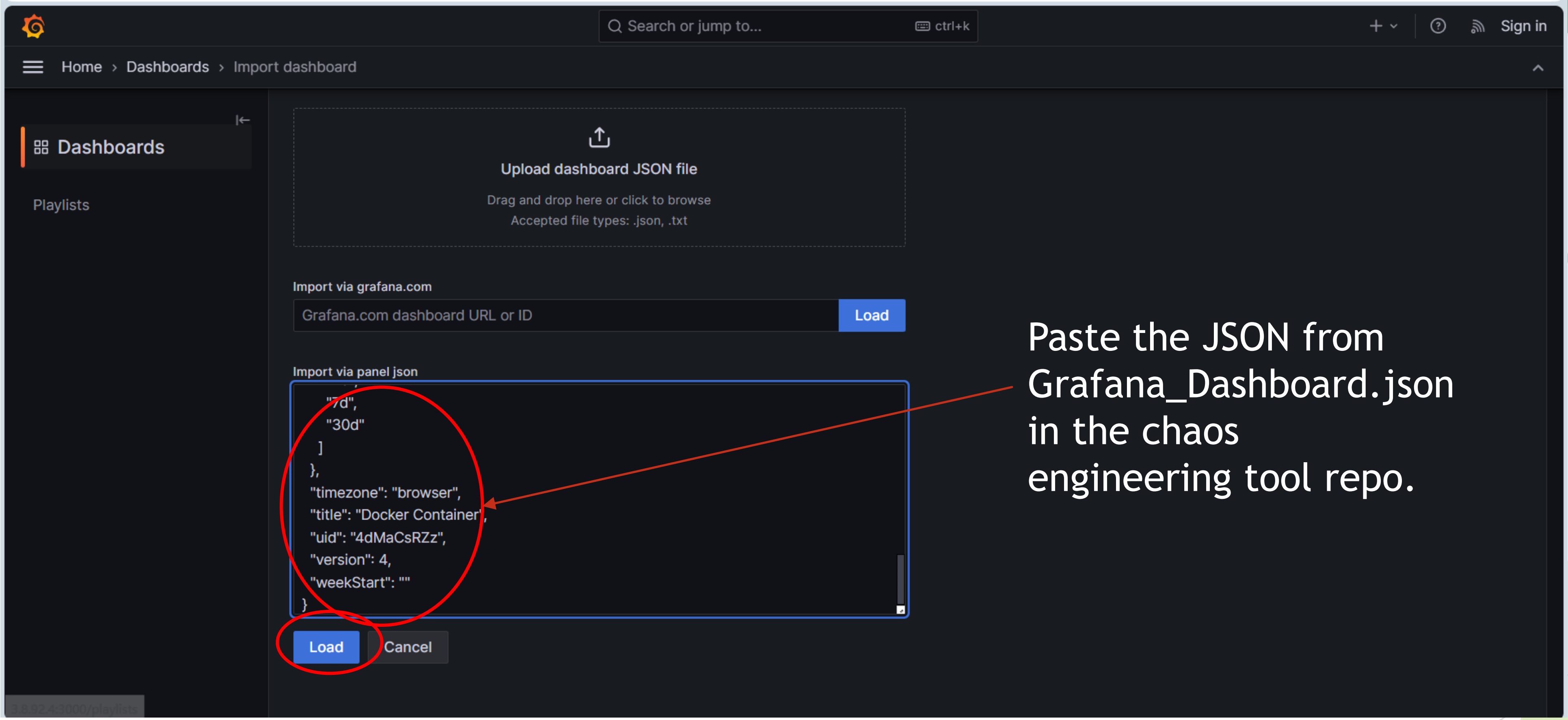

The private IP address of the master node


Search or jump to...
ctrl+k
Sign in
Home › Connections › Data sources › Prometheus
Connections
Add new connection
Data sources
Performance
Prometheus type
Choose
Cache level
Low
Incremental querying (beta)
Disable recording rules (beta)
Other
Custom query parameters
Example: max_source_resolution=5m&timeout
HTTP method
POST
Exemplars
Add
Delete
Save & test

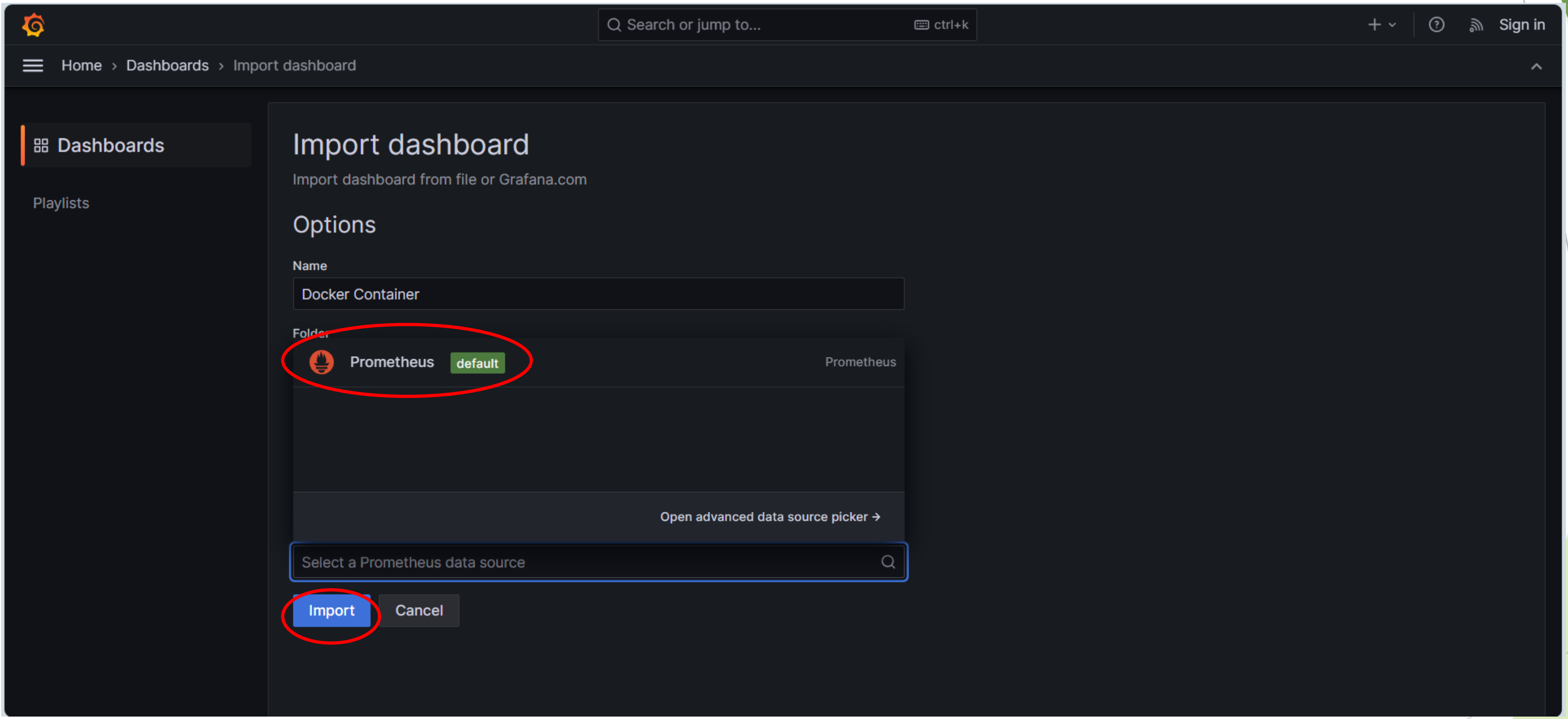


Search or jump to...
ctrl+k
Sign in
Home
Home
Starred
Dashboards
Playlists
Explore
Alerting
Connections
Administration
Need help?
Documentation
Tutorials
Community
Public Slack
Remove this panel
COMPLETE
Add your first data source
Learn how in the docs
DASHBOARDS
Create your first dashboard
Learn how in the docs
Latest from the blog
Aug 31
A better Grafana OnCall: web-based scheduling, mobile app, email support
Does anyone really enjoy being on-call? That looming dread


Search or jump to...
ctrl+k
Sign in
Home
Dashboards
Dashboards
Playlists
Dashboards
Create and manage dashboards to visualize your data
Search for dashboards and folders
New
Filter by tag
Starred
Sort
New dashboard
New folder
Import
General
No results found
3.8.92.4:3000/dashboard/import

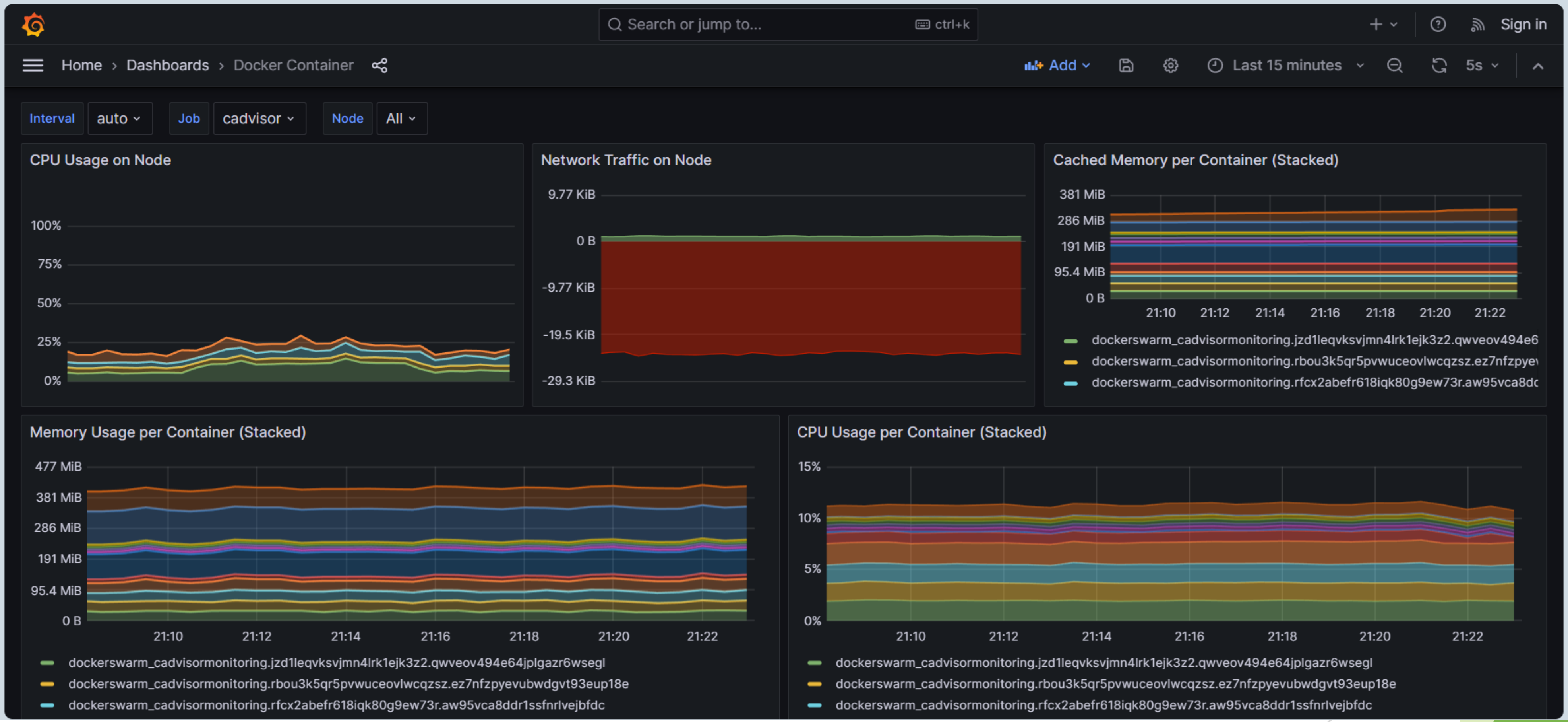



Paste the JSON from Grafana_Dashboard.json in the chaos engineering tool repo.



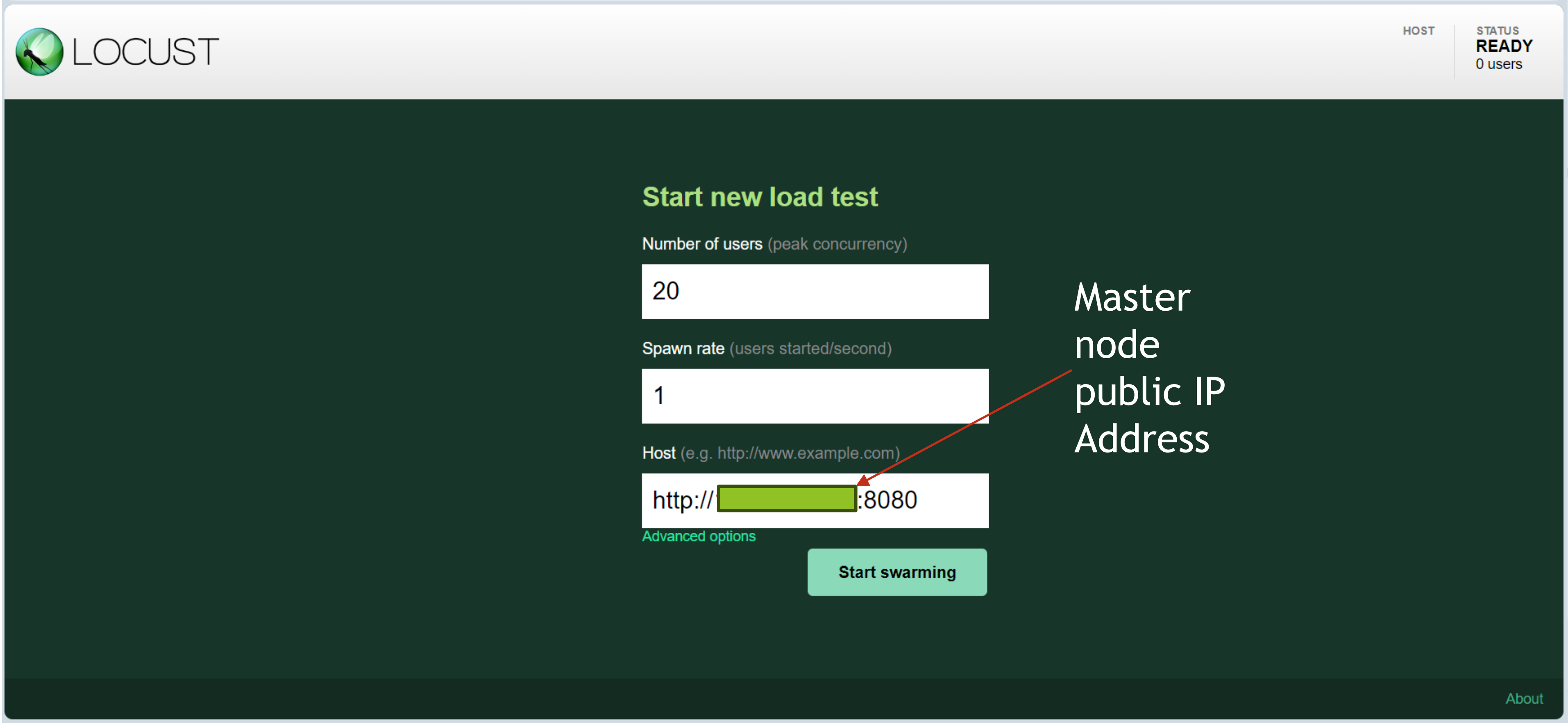
Search or jump to...
ctrl+k
Sign in
Home › Dashboards › Import dashboard
Dashboards
Playlists
Import dashboard
Import dashboard from file or Grafana.com
Options
Name
Docker Container
Folder
Prometheus
default
Prometheus
Open advanced data source picker →
Select a Prometheus data source
Import
Cancel

# This is what the Grafana dashboard should look like (assuming the stack is deployed)

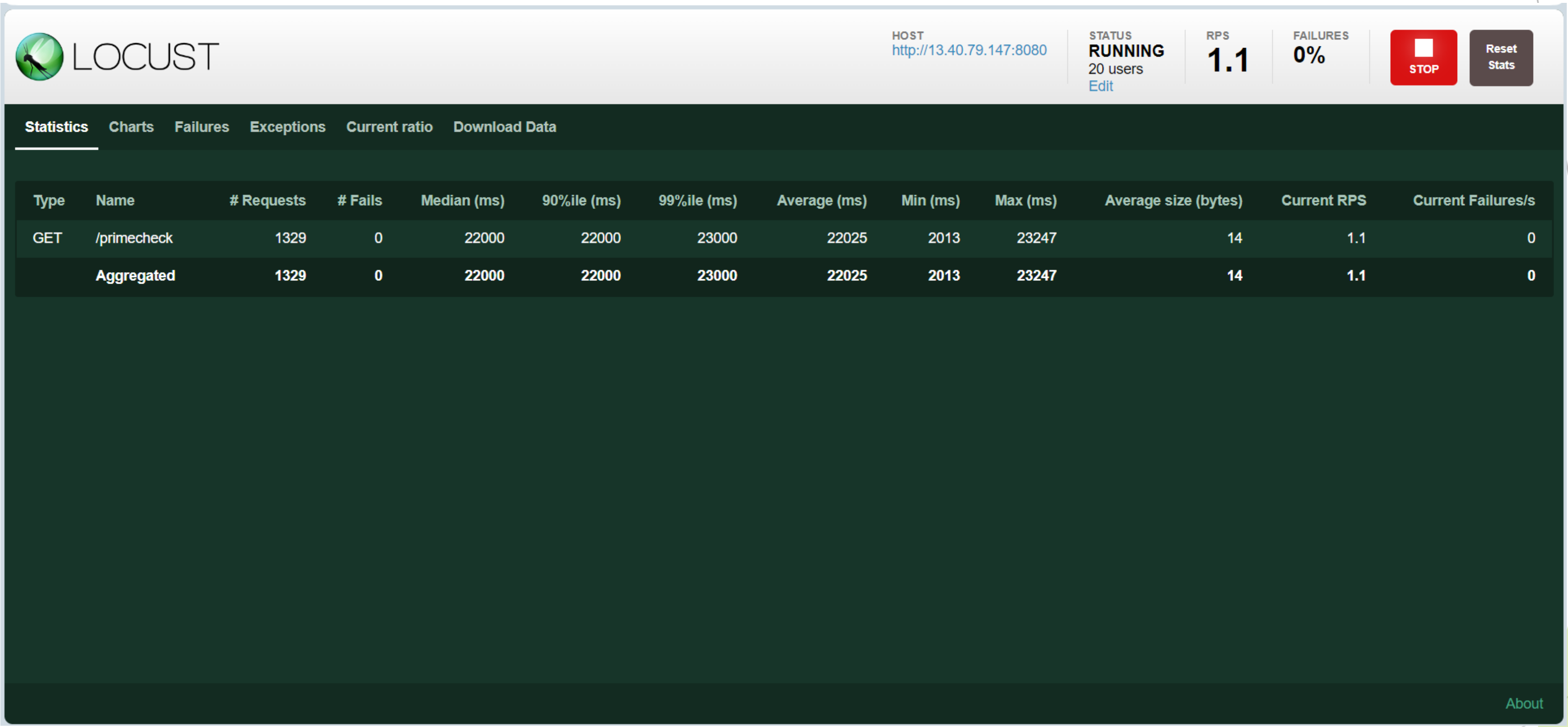


# Running Chaos Experiments – Setting Up Locust

- First, run Locust and monitor the average response time over 5 minutes to define the steady state.
- Locust can be found in the chaos engineering tool repo. Simply ensure docker-compose is installed on your local machine and then inside the locust folder *(cd chaos_engineering_tool/locust)* run the command *docker-compose up*.
- Locust can then be accessed through the web browser at *http://localhost:8089*.
- Locust can then be used to generate HTTP requests (synthetic users).


LOCUST
HOST
STATUS
READY
0 users
Start new load test
Number of users (peak concurrency)
20
Spawn rate (users started/second)
1
Host (e.g. http://www.example.com)
http:// :8080
Advanced options
Start swarming
About
Master node public IP Address

# Screenshot of Locust Running Correctly




| Type | Name | # Requests | # Fails | Median (ms) | 90%ile (ms) | 99%ile (ms) | Average (ms) | Min (ms) | Max (ms) | Average size (bytes) | Current RPS | Current Failures/s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GET | /primecheck | 1329 | 0 | 22000 | 22000 | 23000 | 22025 | 2013 | 23247 | 14 | 1.1 | 0 |
| | **Aggregated** | **1329** | **0** | **22000** | **22000** | **23000** | **22025** | **2013** | **23247** | **14** | **1.1** | **0** |

# Running Chaos Experiments – Defining the Steady State

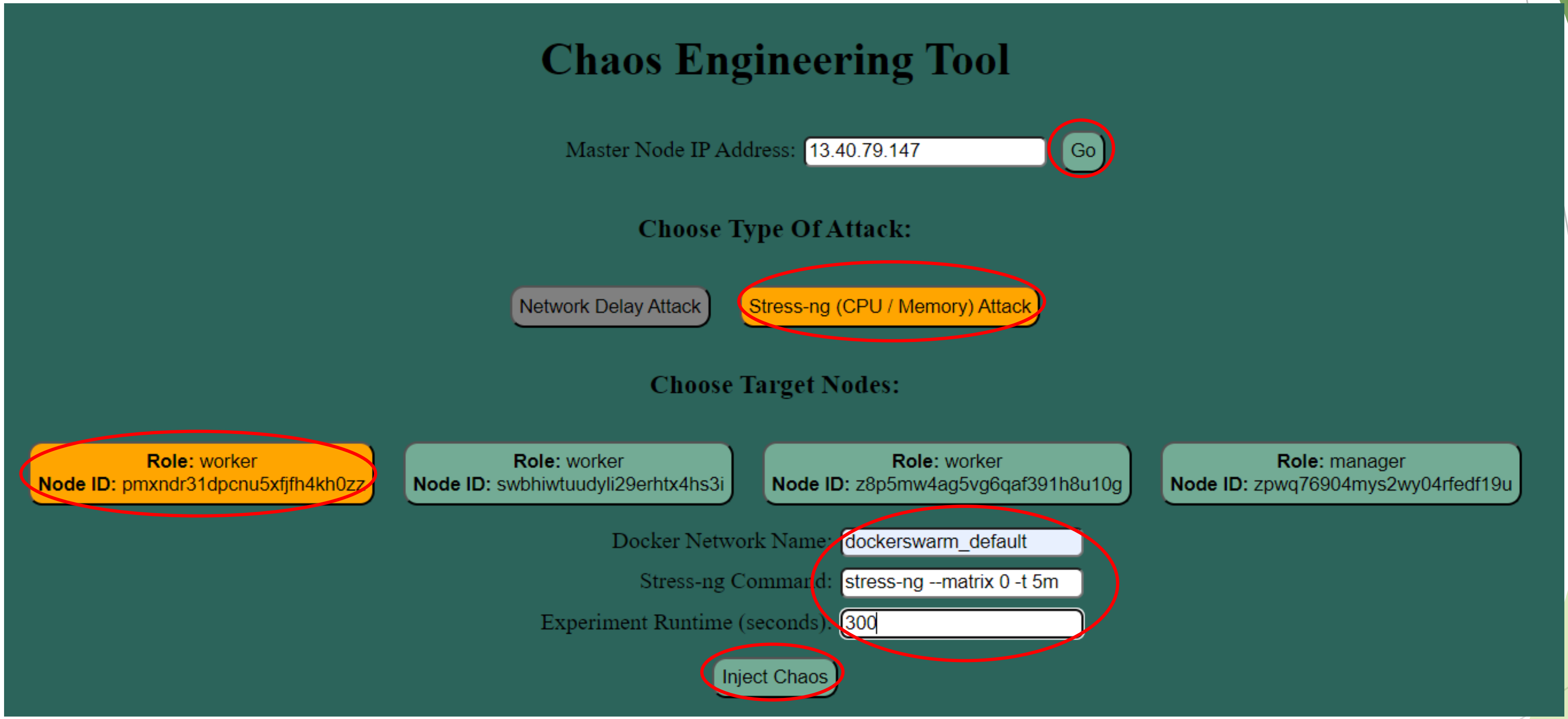
- Run locust for 5 minutes and take note of the system's average response time after 5 minutes.
- The steady state is just the system's normal behaviour. We found a good way to define the steady state in this system is to take the average response time and then add 15% to this value before rounding to 3 significant figures.
- In a real production environment, the steady state could be better defined by monitoring the fluctuations in the response time over a longer time period (it may also vary at different times of the day depending on if the number of users fluctuates).

# Running Chaos Experiments – Set Up the Chaos Tool

1. Assuming the chaos engineering tool has already been cloned from the repo then please ensure npm, React and NodeJS are installed on your local machine.
2. Ensure there are two terminals (three including the one to run locust).
3. Enter the client directory in one terminal and the server terminal in another with the following commands:
   - *cd chaos_engineering_tool/server*
   - *cd chaos_engineering_tool/client*
4. Run *npm install* inside both the client and server folder.
5. Run *node index.js* in the server folder
6. Run *npm start* in the client folder

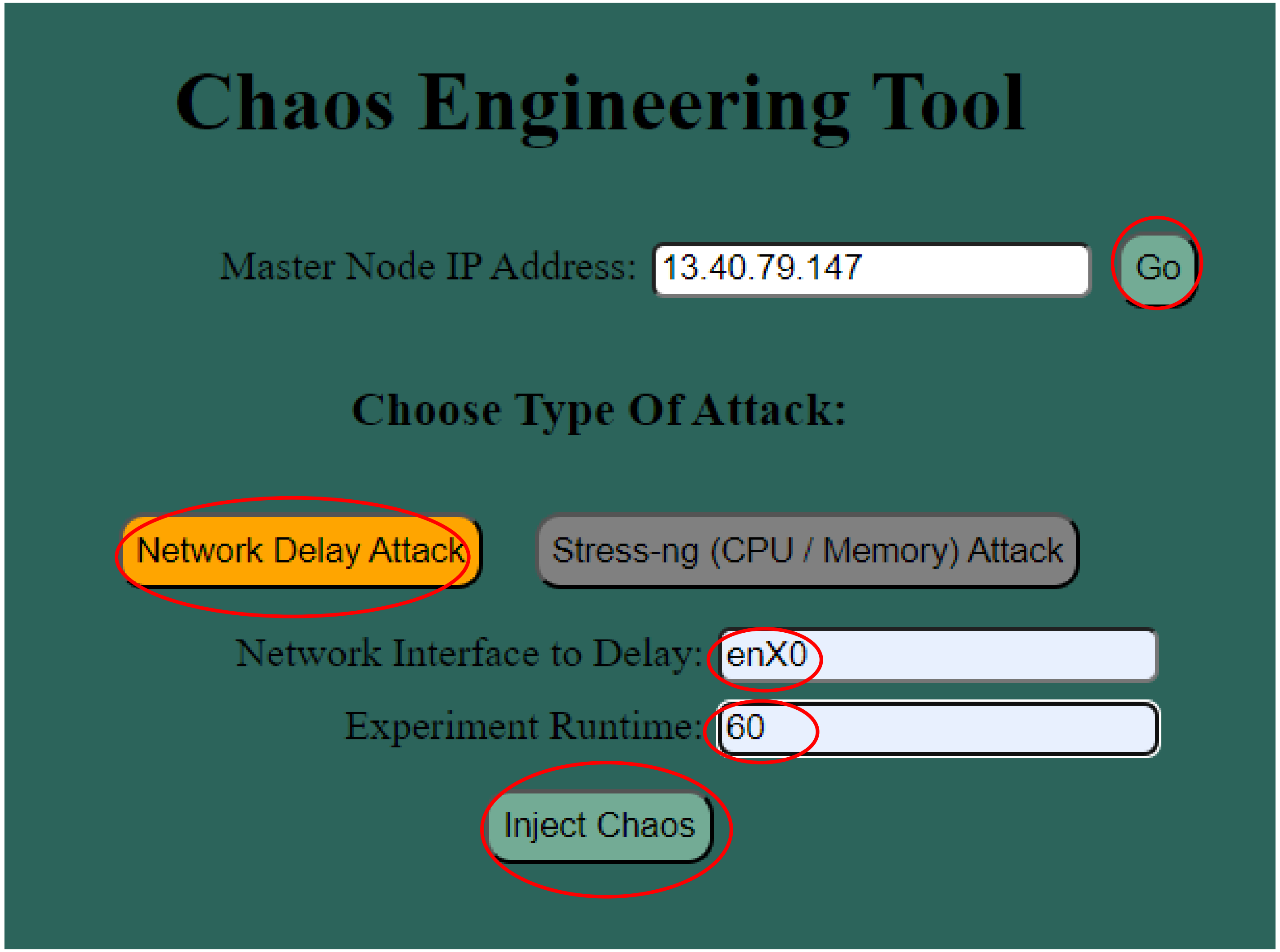
# Running Chaos Experiments – Use the Chaos Tool for a Stress-ng Attack

- The tool should load automatically in your default web browser but if not, it can be located at the address: *http://localhost:3005*
- The screenshot that follows shows a matrix attack. This attack can also be swapped for a virtual memory attack by using a command such as *stress-ng –vm 0 –t 5m*
- Note: The zero simply means to attack all the CPU cores, this can be altered to attack a specific number of CPU cores if desired. Further information on stress-ng commands can be found here - https://manpages.ubuntu.com/manpages/xenial/man1/stress-ng.1.html
- These parameters can be modified as desired, the experiment runtime can be changed, the command used or the experiment runtime
- Note: It is important that the experiment runtime field and the experiment runtime specified in the stress-ng command are the same.
- When all the parameters are set simply run Locust and the chaos experiment at the same time and monitor the average response time to see if it falls outside of the system's steady state. Grafana can also be used to monitor the system metrics.


Chaos Engineering Tool
Master Node IP Address:
13.40.79.147
Go
Choose Type Of Attack:
Network Delay Attack
Stress-ng (CPU / Memory) Attack
Choose Target Nodes:
Role: worker
Node ID: pmxndr31dpcnu5xfjfh4kh0zz
Role: worker
Node ID: swbhiwtuudyli29erhtx4hs3i
Role: worker
Node ID: z8p5mw4ag5vg6qaf391h8u10g
Role: manager
Node ID: zpwq76904mys2wy04rfedf19u
Docker Network Name:
dockerswarm_default
Stress-ng Command:
stress-ng --matrix 0 -t 5m
Experiment Runtime (seconds):
300
Inject Chaos

# Running Chaos Experiments – Use the Chaos Tool for a Network Attack

- The attack can be configured as shown on the next slide
- The experiment runtime can be modified as desired
- The interface to delay should be enX0.
- If this is not the case, it may be eth0 instead (or similar)
- The networks on an instance can be checked with the command *“ip a”*
- Targeting enX0 will delay outgoing traffic, but it will also affect communication between nodes in the docker swarm. It will be necessary to allow for this when performing the chaos experiments (e.g., if the delay added is 500ms, then allow for this additional response time, when monitoring the steady state)
- Once again, when all the parameters are set simply run Locust and the chaos experiment at the same time and monitor the average response time to see if it falls outside of the system’s steady state. Grafana can also be used to monitor the system metrics


Chaos Engineering Tool
Master Node IP Address:
13.40.79.147
Go
Choose Type Of Attack:
Network Delay Attack
Stress-ng (CPU / Memory) Attack
Network Interface to Delay:
enX0
Experiment Runtime:
60
Inject Chaos

# Running Chaos Experiments – How to Check the Network Latency has been Injected

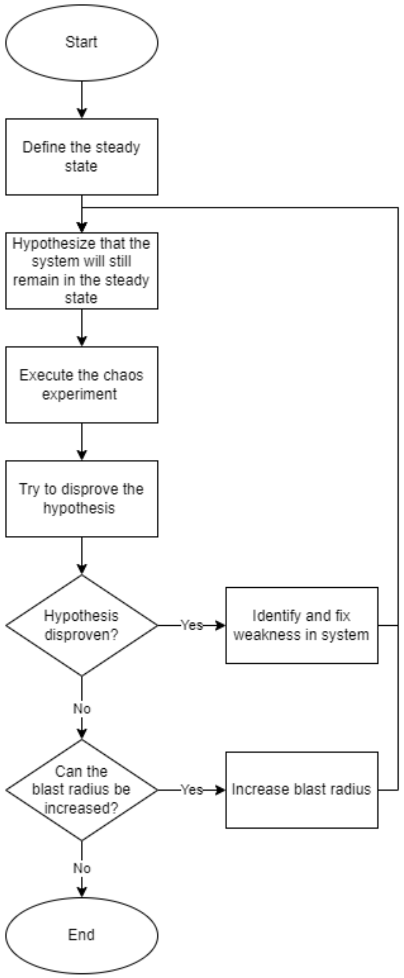
1. Get the names of all the running tasks on the node using *“docker ps”*
2. Use the command *docker exec –it [container id] sh* on any container (apart from the chaos engineering container)
3. From inside this container ping the ID of another container running on another node.

- When the network latency is added, there will be an increase of 2x what the delay inject was (because it is a bidirectional delay)
- *Ping 8.8.8.8* can also be used to ping Google’s DNS server and check the delay on the outgoing traffic
- An example of how to ping another container is shown on the next slide

```
aws   Services   Search   [Alt+S]                                   London ▾   Travis200 ▾
[ec2-user@ip-172-31-41-147 ~]$ docker ps
CONTAINER ID   IMAGE                              COMMAND                  CREATED       STATUS                 PORTS      NAMES
c4d1ca880249   prom/prometheus:latest             "/bin/prometheus --c…"   5 hours ago   Up 5 hours             9090/tcp   dockerswarm_prometheus.zpwq76904mys2wy04rfedf19u.icvf2c0hcukuizpz
uti70kbzw
96b5d28a657f   grafana/grafana-enterprise:latest  "/run.sh"                5 hours ago   Up 5 hours             3000/tcp   dockerswarm_grafana.1.k9hbrys60aoe3anxmbobsbc75
fc72c545dec3   gcr.io/cadvisor/cadvisor:v0.45.0   "/usr/bin/cadvisor -…"   5 hours ago   Up 5 hours (healthy)   8080/tcp   dockerswarm_cadvisormonitoring.zpwq76904mys2wy04rfedf19u.xkswpkl0
v8o0k1sgcj2m29zsp
8dd3d518ccd2   dockersamples/visualizer:latest    "/sbin/tini -- node …"   5 hours ago   Up 5 hours (healthy)   8080/tcp   dockerswarm_dockerswarmvisualizer.1.8j7dlasx518slzft7dwgi1ipm
b9584d61c6ef   prom/node-exporter:latest          "/bin/node_exporter …"   5 hours ago   Up 5 hours             9100/tcp   dockerswarm_node-exporter.zpwq76904mys2wy04rfedf19u.w35pju69ms4v8
4fxyj1w6gc49
[ec2-user@ip-172-31-41-147 ~]$ docker exec -it fc72c545dec3 sh
/ # ping [redacted]
PING 94a845d47db3 (10.0.1.32): 56 data bytes
64 bytes from 10.0.1.32: seq=0 ttl=127 time=0.622 ms
64 bytes from 10.0.1.32: seq=1 ttl=127 time=0.647 ms
64 bytes from 10.0.1.32: seq=2 ttl=127 time=0.564 ms
64 bytes from 10.0.1.32: seq=3 ttl=127 time=0.555 ms
64 bytes from 10.0.1.32: seq=4 ttl=127 time=0.606 ms
64 bytes from 10.0.1.32: seq=5 ttl=127 time=0.684 ms
64 bytes from 10.0.1.32: seq=6 ttl=127 time=0.622 ms
64 bytes from 10.0.1.32: seq=7 ttl=127 time=0.749 ms
64 bytes from 10.0.1.32: seq=8 ttl=127 time=1.582 ms
64 bytes from 10.0.1.32: seq=9 ttl=127 time=0.614 ms
64 bytes from 10.0.1.32: seq=10 ttl=127 time=0.526 ms
64 bytes from 10.0.1.32: seq=11 ttl=127 time=0.617 ms
64 bytes from 10.0.1.32: seq=12 ttl=127 time=0.534 ms
64 bytes from 10.0.1.32: seq=13 ttl=127 time=0.647 ms
64 bytes from 10.0.1.32: seq=14 ttl=127 time=1000.982 ms
64 bytes from 10.0.1.32: seq=15 ttl=127 time=1000.777 ms
64 bytes from 10.0.1.32: seq=16 ttl=127 time=1000.655 ms
64 bytes from 10.0.1.32: seq=17 ttl=127 time=1000.705 ms
```

Container ID running on another node

# Applying the Chaos Engineering Methodology to this Set Up with Stress-ng Faults




1. Run Locust for 5 minutes and use then from this to determine the upper bound steady-state value (add 15% to the average response time and round to 3 significant figures).
2. Hypothesise that the system will remain in its steady state. For example: *injecting an X stressor fault into N nodes of the Docker Swarm will not cause the system's behaviour to change from its steady state. The average response time of the Java Benchmark App will remain under the steady-state threshold.*
3. Execute the Chaos Experiment.
4. Try to disprove the hypothesis (monitor if the average response time goes above the steady state).
5. If the hypothesis has been disproven, then increase the number of replicas of the Java Benchmark App and go back to step 2.
6. If the blast radius can be increased (not all nodes are being targeted except the master node), then increase the blast radius and go back to step 2.
7. If the blast radius cannot be increased (all nodes except the master node are targeted) and the hypothesis has not been disproven, then we can now have increased confidence in the system's resilience to the fault that was tested.

# Pros and Cons of this Experiment Set Up

**Cons**

- Cannot target master node due to Grafana being hosted on here.
- Is not a true production environment so it is necessary to generate synthetic user requests with Locust.
- Targeting enX0 for network attacks also affects outgoing network traffic.

**Pros**

- Enables us to practice chaos engineering and following a methodology.
- Can make simple alterations (increase the java benchmark app replicas) to fix the “faults” identified.
- Grafana visualisations make monitoring technical metrics possible.
- Proof of concept for how chaos engineering can be beneficial.

# References

[1] “Chaos Monkey at Netflix: the Origin of Chaos Engineering,” accessed: 02/09/2023. [Online]. Available: https://www.gremlin.com/chaos-monkey/the-origin-of-chaos-monkey/

[2] “What is chaos engineering? Chaos engineering and its principles explained,” accessed: 02/09/2023. [Online]. Available: https://www.techtarget.com/searchitoperations/definition/chaos-engineering

[3] “PRINCIPLES OF CHAOS ENGINEERING,” accessed: 02/09/2023. [Online]. Available: https://principlesofchaos.org/