Cloud Computing final project

A cloud-based file storage system and the OSU benchmarks in k8s

Isac Pasianotto

2023-02-12

Cloud **basic** module

Cloud **basic** module

1. Cloud **basic** module
   - Requirements
   - Nextcloud: the chosen solution
   - Deployed structure
   - Load test: Locust
   - Scalability
2. Cloud **advanced** module: Nextcloud in a kubernetes flavor
3. Cloud **advanced** module: OSU benchmarks in k8s

Requirements

Identify, deploy and implement a cloud-based file storage system

- User / admin roles
- User should be able to login, upload, download, delete files, have a personal folder
- Admin should be able to manage users, quotas, etc
- Security evaluation
- Scalability evaluation
- Test the system with a load test

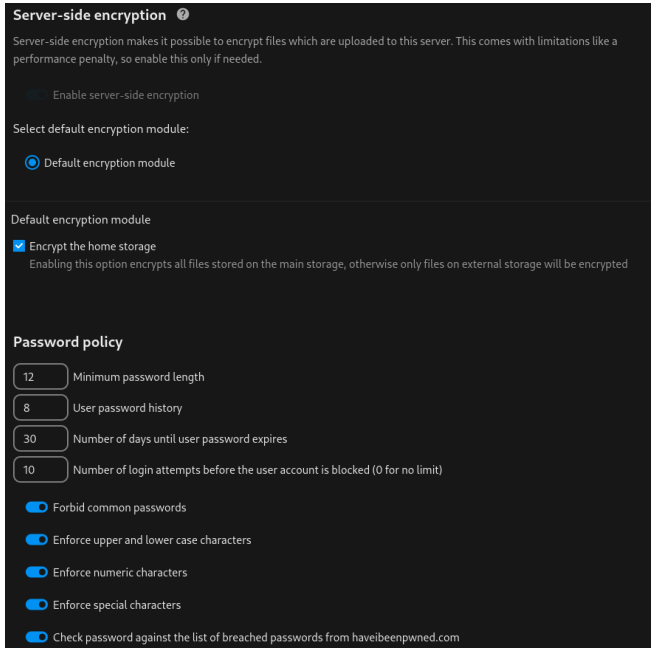
Nextcloud: the chosen solution

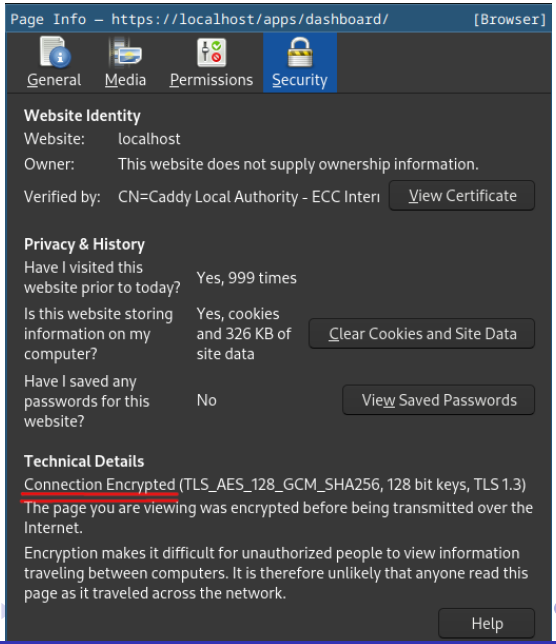
- Comes with already built-in all the required features (User/admin, download/upload, private folder, etc)
- Has a large community and a lot of plugins
- Can be easily deployed using docker (already available image)
- Supports different types of storage (local, NFS, S3, etc)
- Supports various database backends (MySQL, PostgreSQL, SQLite)

Nextcloud: the chosen solution (con'd)

- Possibility to enable **server-side encryption**[a]
- Customizable **password minimum requirements** and **password policy**
- **Brute-force protection** (additional app)
- **Two-factor authentication**
- Accepts **OAuth2** external authentication

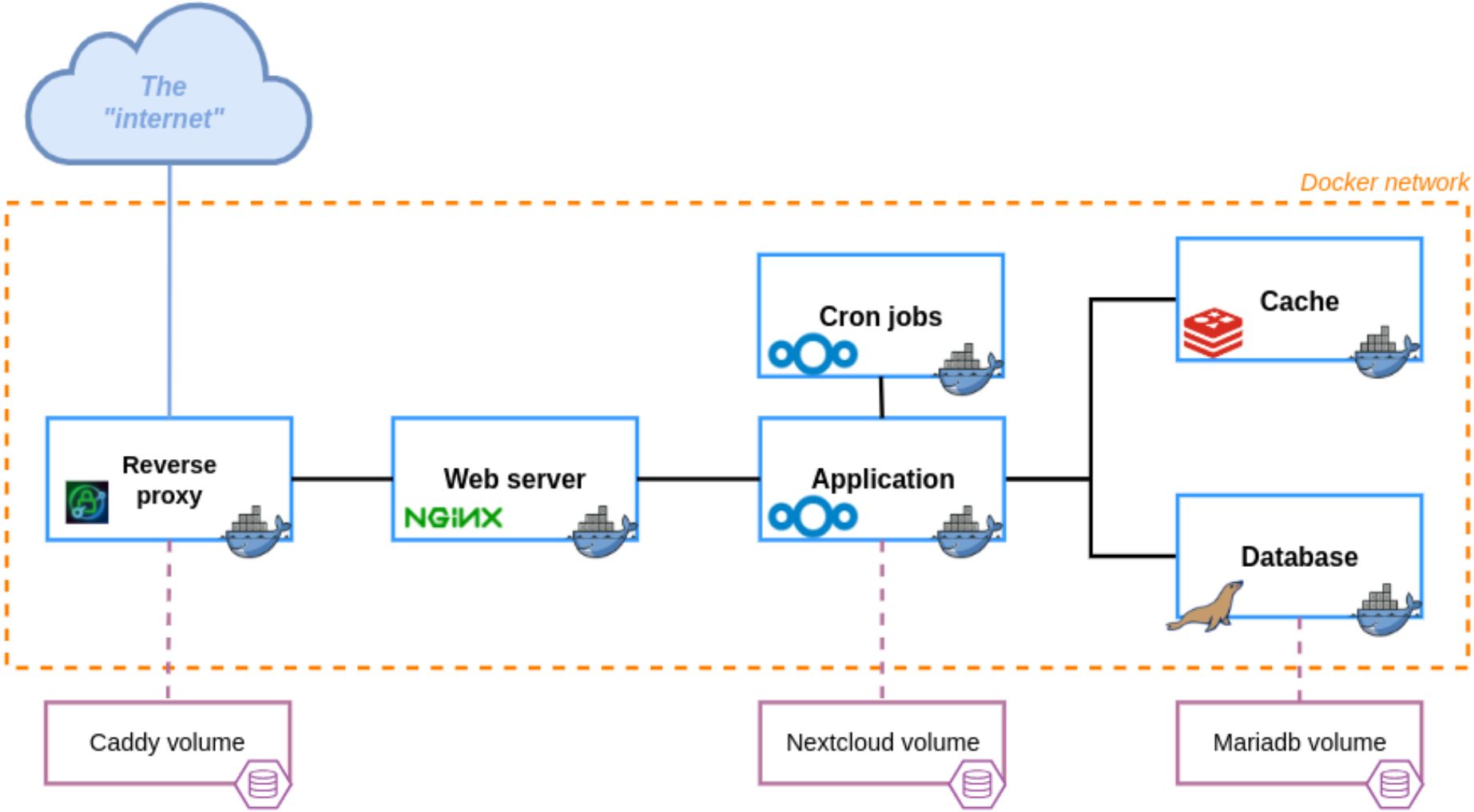
And what about the connection user-server?
⇒ **Caddy** automatically provides a **Let's Encrypt** certificate





[a] Drawback: more CPU usage

Deployed structure



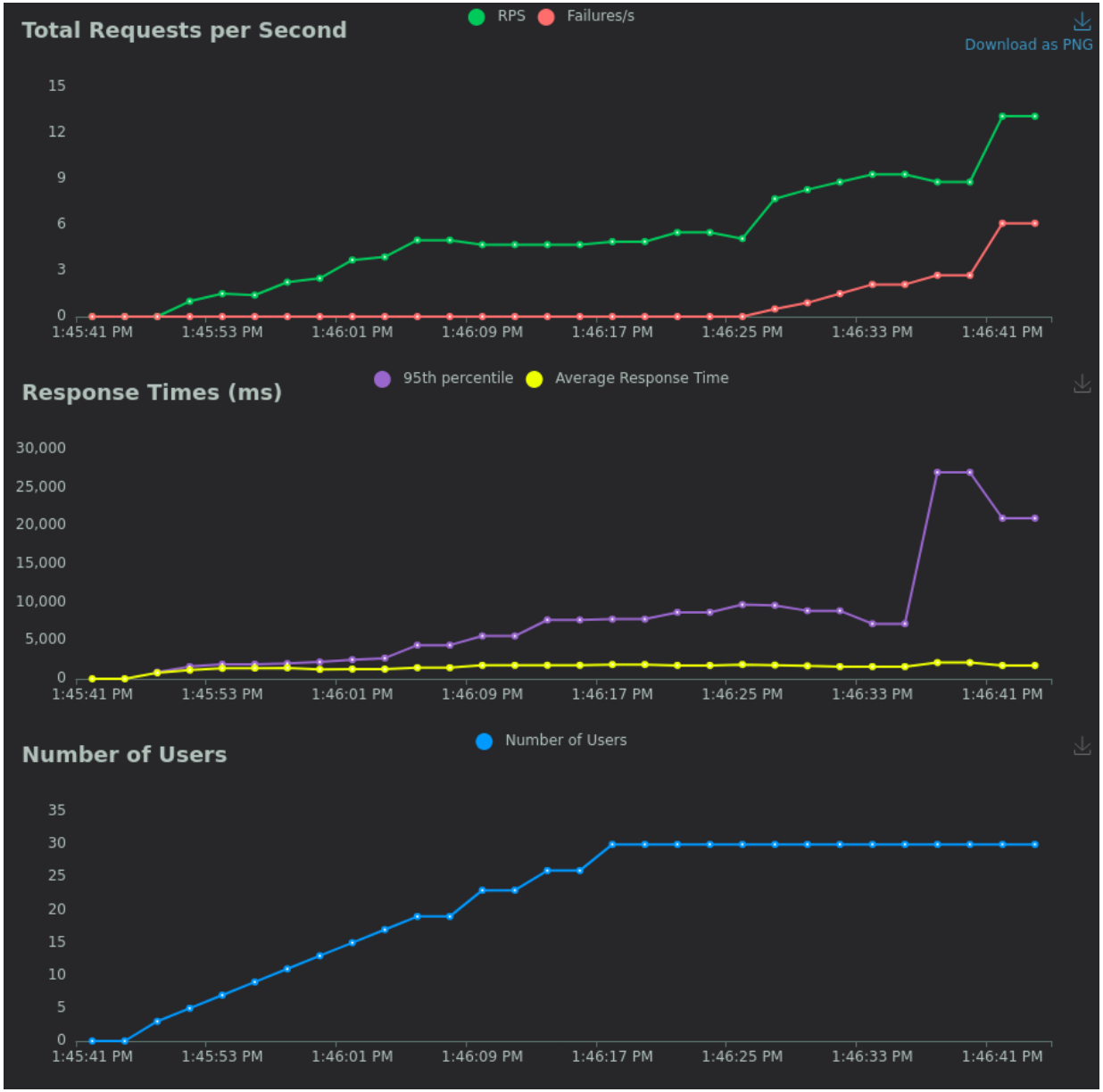
Load test: Locust

Locust is an open-source load testing tool, which allows defining user behavior with Python code.

- start with 1 user
- add 1 user every second until a predefined quota is reached
- every second, each active user tries to perform a random task
- each user has a wait time between 0.5 and 3 seconds before declaring the task as failed

Tasks	%
upload_small	30%
upload_medium	35%
upload_large	5%
download	30%

Load test: Locust (con'd)



Load test performed with server-side encryption enabled. During the test, I got warnings from the server about the CPU usage over 90%.

Scalability

I've considered all the possible solutions in the case of a real-world scenario

- **On-site** solution: Build on a owned server, with appropriate hardware
- **Infrastructure as a Service** (IaaS): rent needed resources from a cloud provider, but manage everything by yourself
- **Platform as a Service** (PaaS): rent a pre-configured environment, and focus only on the application and data
- **Software as a Service** (SaaS): rent a ready-to-use application, just use it with your data

Scalability (con'd)

	on-site	*IaaS*	*PaaS*	*SaaS*
hardware acquisition/ cost	high	null	null	null
maintenance costs	high	low	null	null
predictable costs	high	medium	low	low
Renting costs	null	low	medium	high
Need to care about security	high	medium	on your app	trust the vendor
Physically own the data	yes	no	no	no
Scalability	completely on you	rent more resources	pay more	pay more
Time to get ready	high	medium	low	almost null

Cloud **advanced** module: Nextcloud in a kubernetes flavor

Cloud **advanced** module: Nextcloud in a kubernetes flavor

1. Cloud **basic** module
2. Cloud **advanced** module: Nextcloud in a kubernetes flavor
   - Requirements
   - Different architecture
   - How it works
   - Benefits of kubernetes
3. Cloud **advanced** module: OSU benchmarks in k8s

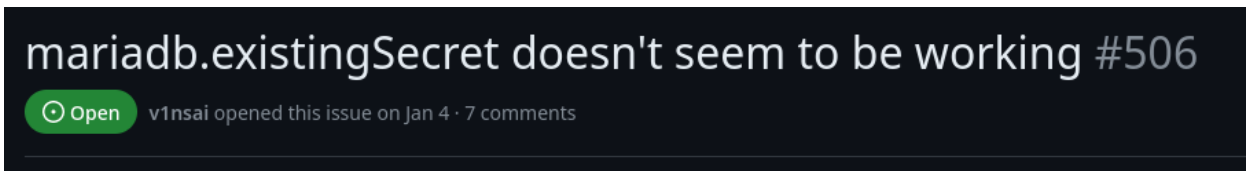
Requirements

Re-deploy the solution of the previous exercise, but in a kubernetes environment

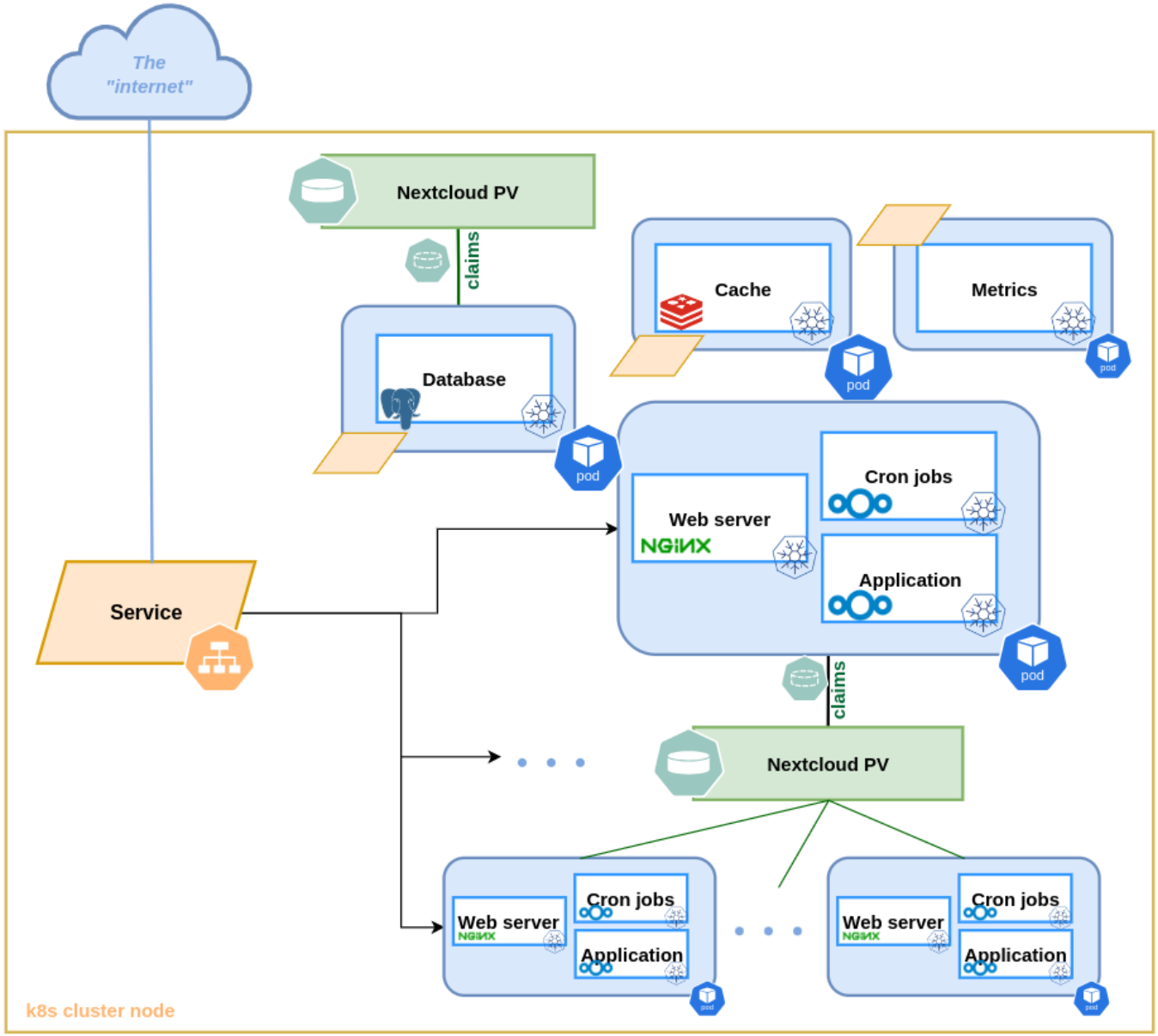
- The cluster must run `k8s`, one node is required
- The volume must survive a `pod crash` and `accidental deletion`
- The service must be accessible from the user via IP or FQDN
- Eventual databases or **third-party services** necessary for the deployed software must **run in their pod**

Different architecture

- One node $\rightarrow$ the control plane is also the worker node
- Used the nextcloud `helm chart`
- Used `metallb` to expose the service
- Access is done through a `service`
- All the pods run in a dedicated namespace
- The kubernetes orchestrator automatically takes care of the pods, respawning them if needed

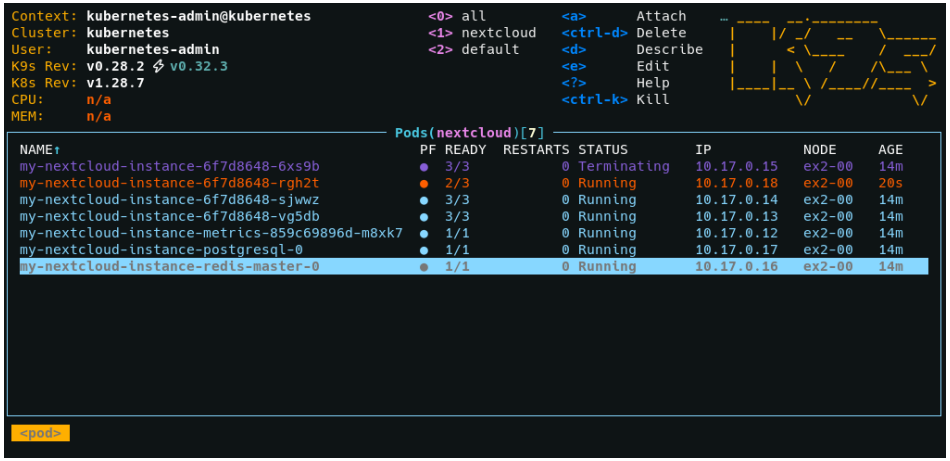


Different architecture (con'd)



How it works

- Installation done with `helm` and a custom `values.yaml` file
- Manually created `persistent volume` and `persistent volume claim`
- Manually created `secrets` for storing credentials (avoiding them to be stored in the `values.yaml` file)
- Postgres, Redis and metrics run in their own pods
- System can survive a pod crash or accidental deletion



After a pod deletion, the orchestrator automatically respawns it

```
[vagrant@ex2-00 nextcloud-helm&yaml]$ kubectl get service -n nextcloud
NAME                                TYPE           CLUSTER-IP       EXTERNAL-IP       PORT(S)          AGE
my-nextcloud-instance               LoadBalancer   10.102.241.179   192.168.121.200   8080:32744/TCP   12m
my-nextcloud-instance-metrics       LoadBalancer   10.96.133.84     192.168.121.201   9205:31045/TCP   12m
my-nextcloud-instance-postgresql    ClusterIP      10.97.249.153    <none>            5432/TCP         12m
my-nextcloud-instance-postgresql-hl ClusterIP      None             <none>            5432/TCP         12m
my-nextcloud-instance-redis-headless ClusterIP     None             <none>            6379/TCP         12m
my-nextcloud-instance-redis-master  ClusterIP      10.101.68.84     <none>            6379/TCP         12m
[vagrant@ex2-00 nextcloud-helm&yaml]$ curl -s 192.168.121.200:8080 | head -n 15
<!DOCTYPE html>
<html class="ng-csp" data-placeholder-focus="false" lang="en" data-locale="en" translate="no" >
        <head
 data-requesttoken="vd9b0WNSIosT2uf2TM3D+xuRQq4FnT+vMQLU2Su6Hp8=:5Ls/mgVmYP53t7aPL6P6i1HHEupXzln9HkiDrG+VXMY=">
                <meta charset="utf-8">
                <title>
                        Nextcloud                </title>
                <meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0">
                                <meta name="apple-itunes-app" content="app-id=1125420102">
                                <meta name="theme-color" content="#0082c9">
                <link rel="icon" href="/core/img/favicon.ico">
                <link rel="apple-touch-icon" href="/core/img/favicon-touch.png">
                <link rel="mask-icon" sizes="any" href="/core/img/favicon-mask.svg" color="#0082c9">
                <link rel="manifest" href="/core/img/manifest.json" crossorigin="use-credentials">
                <link rel="stylesheet" href="/apps/theming/css/default.css?v=70e2b24f-0">
```

Benefits of kubernetes

Some of the benefits of using kubernetes as container orchestrator are:

- **Availability**: automatic pod respawning in case of crash
- **Scalability**: easy to add more nodes, and the `Horizontal Pod Autoscaler` can automatically add more pods
- **Topology awareness**: easy to schedule pods on different nodes to avoid single point of failure
- **Resource management**: kubernetes can automatically manage the resources of the pods, setting limits to the CPU and memory usage
- **Portability**: moving the configuration to another cluster/cloud provider should be easy

Cloud **advanced** module: OSU benchmarks in k8s

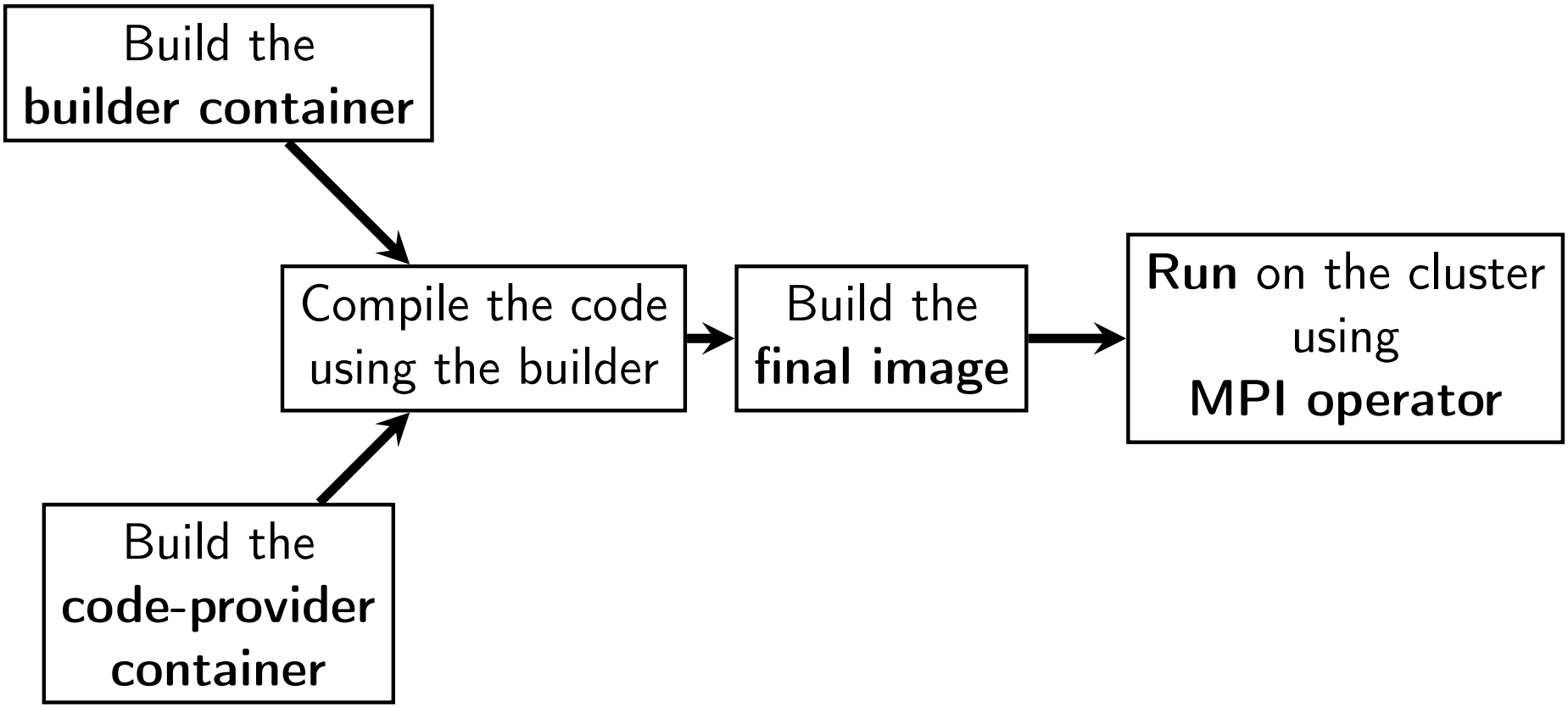
Cloud **advanced** module: OSU benchmarks in k8s

1. Cloud **basic** module
2. Cloud **advanced** module: Nextcloud in a kubernetes flavor
3. Cloud **advanced** module: OSU benchmarks in k8s
   - Requirements
   - Building the image
   - Calico vs Flannel
   - results

Requirements

- 2 nodes `k8s` cluster
- Nodes must talk with either `Calico` or `Flannel`
- `MPI` operator
- Perform the `OSU` benchmark to evaluate the latency between the nodes

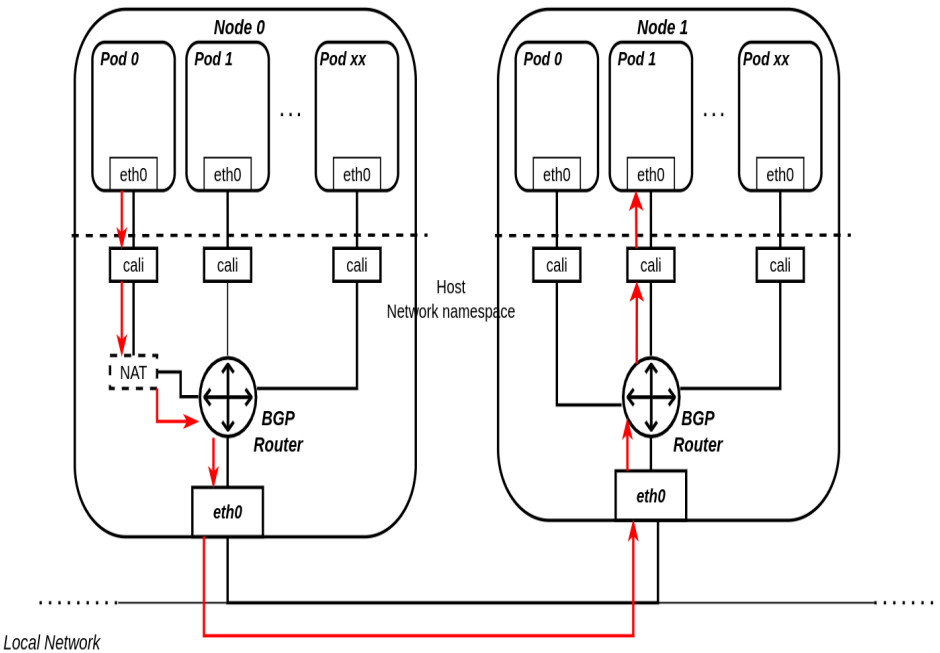
Building the image

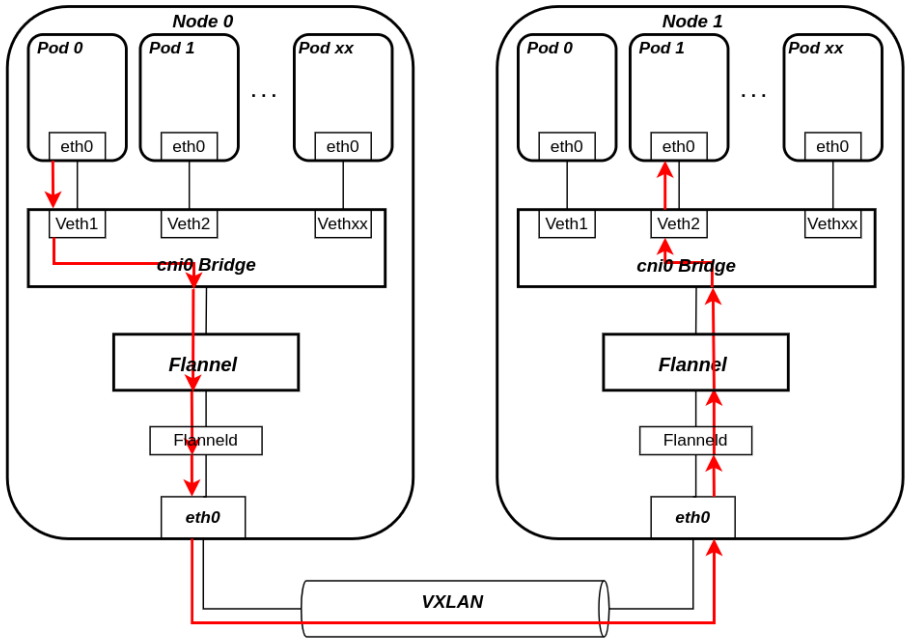


No need to reinvent the wheel:

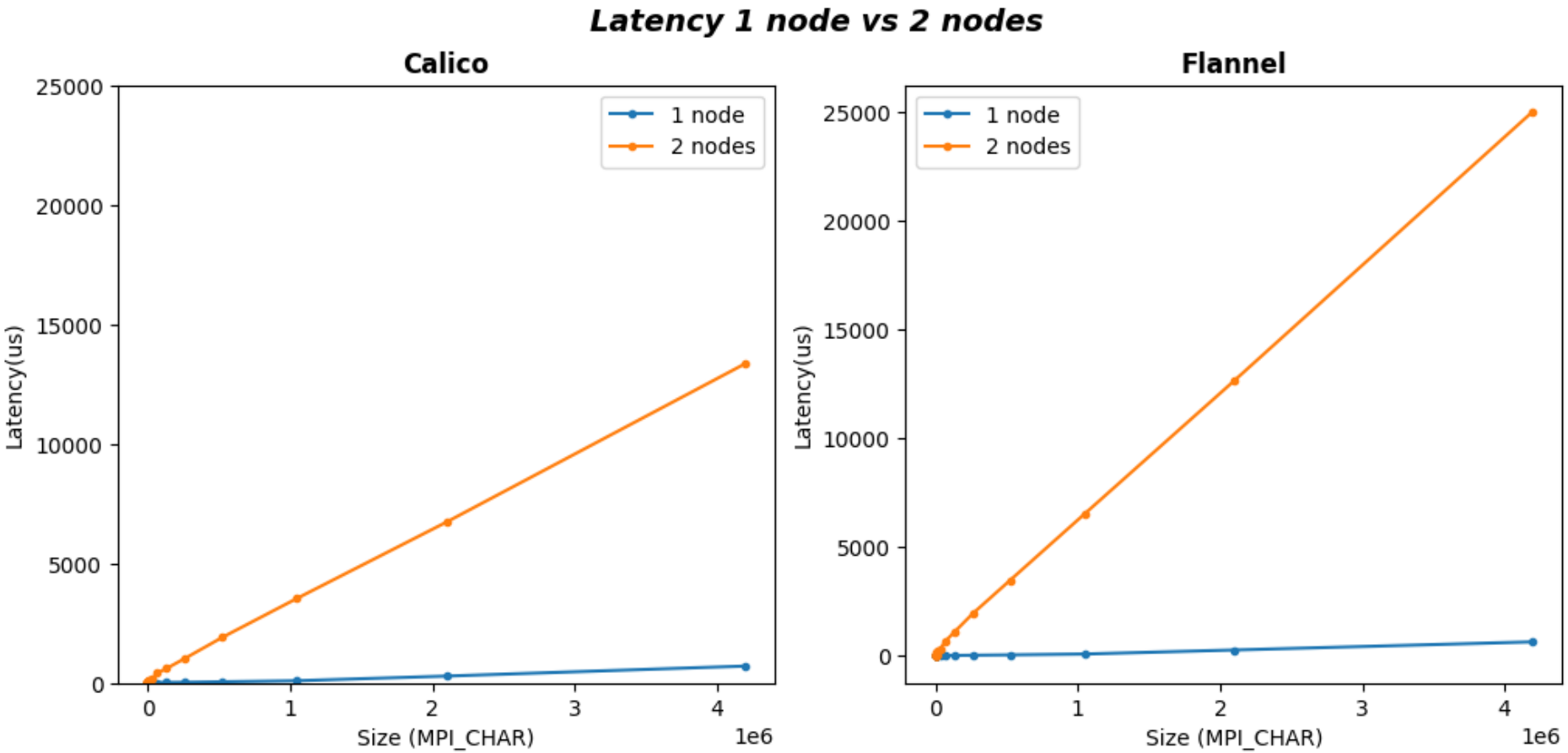
Calico vs Flannel

From pod 0 (node 0) to pod 1 (node 1):



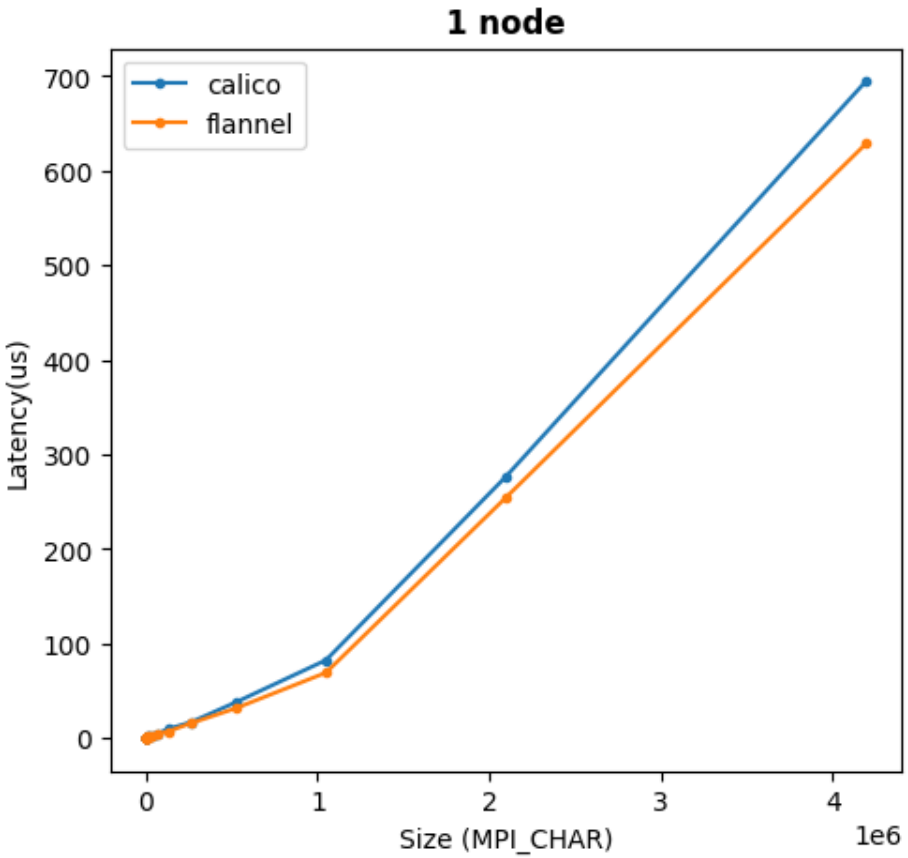


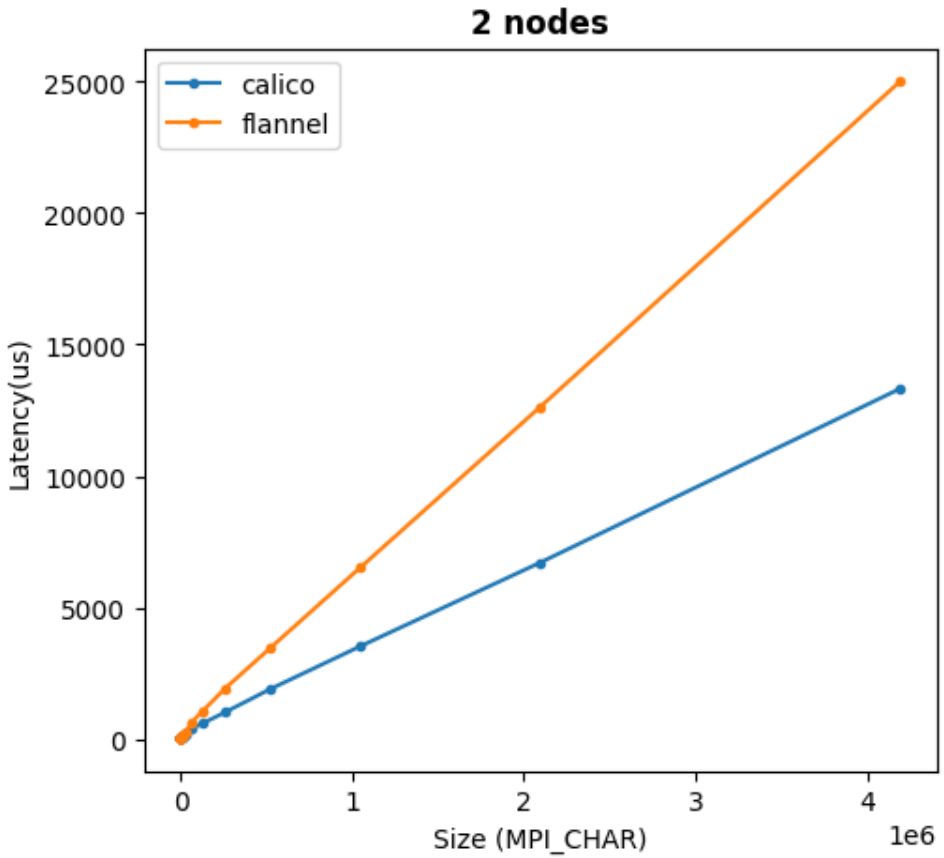
Results[1]



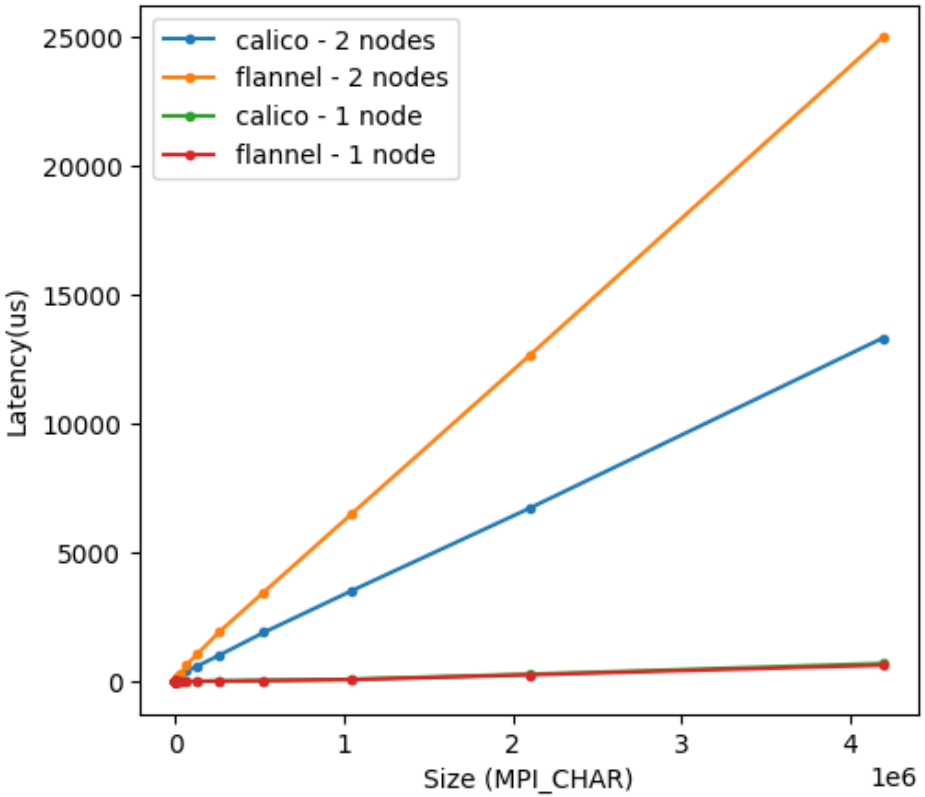
[1] Each point represents the average of 20 runs.

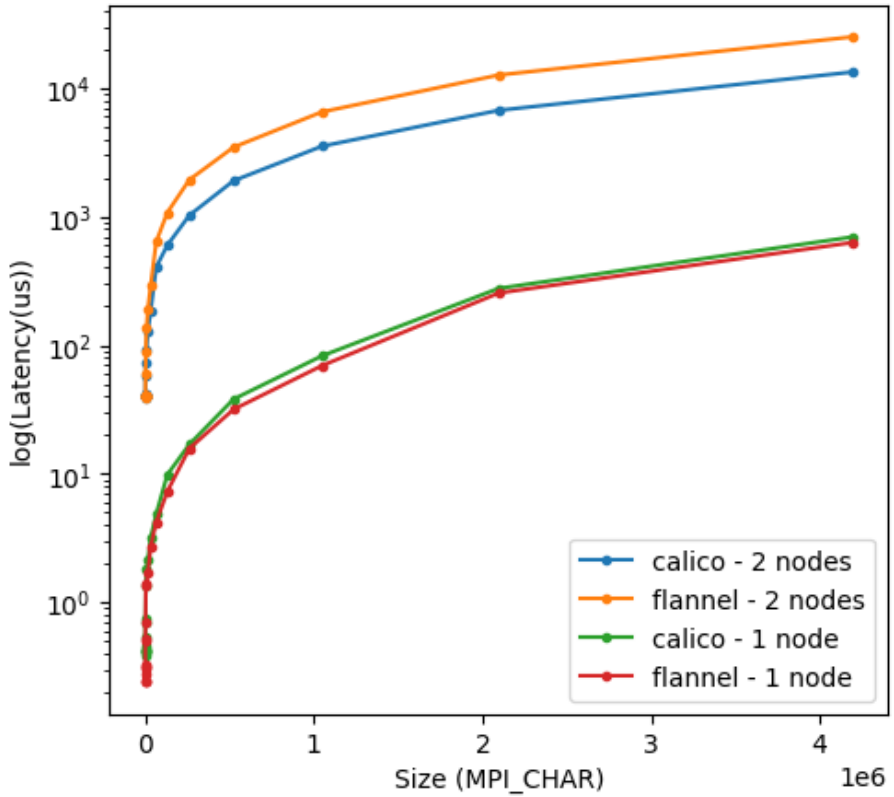
Results (con'd)





Results (con'd)





Thank you for your attention