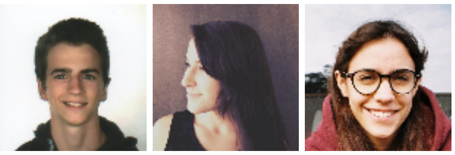
LAB-05-Report-ist189399.md

LAB04 README

AGISIT 2021-2022

Authors

Team 20A

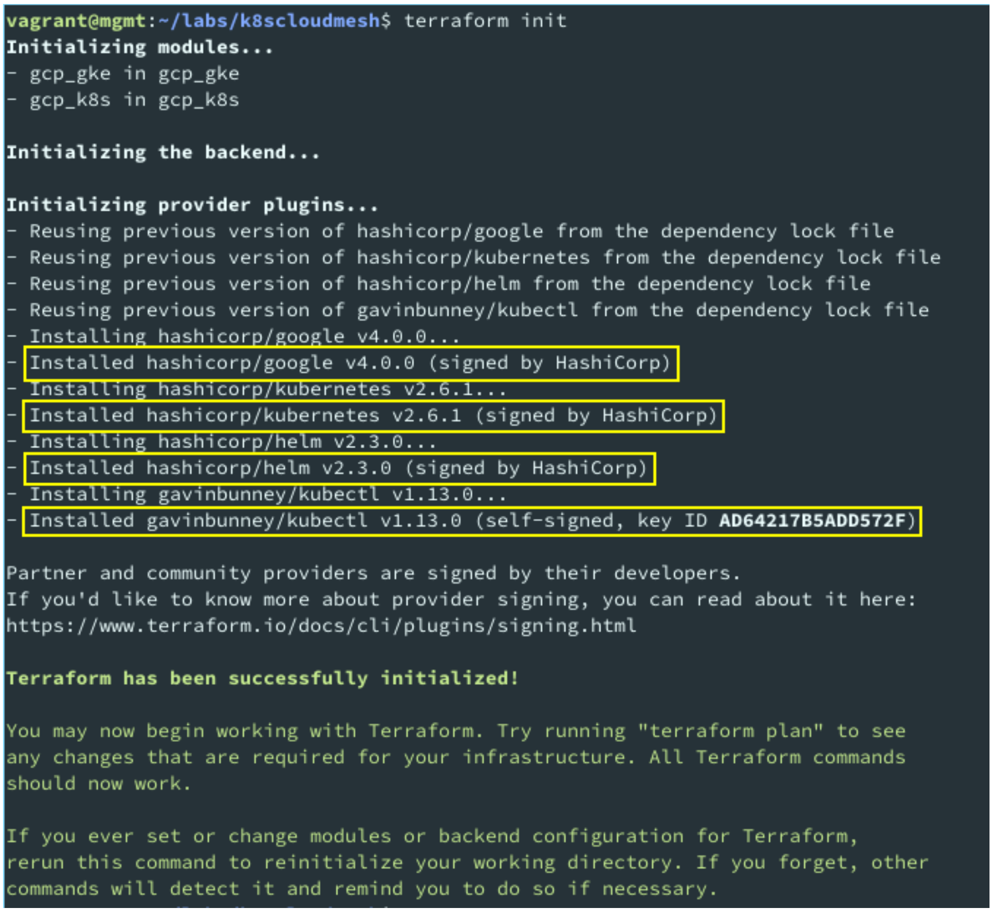


Number	Name	Username	Email
ist189399	Afonso Goncalves	https://git.rnl.tecnico.ulisboa.pt/ist189399	mailto:afonso.corte-real.goncalves@tecnico.ulisboa.pt
ist190621	Maria Filipe	https://git.rnl.tecnico.ulisboa.pt/ist190621	mailto:maria.j.d.c.filipe@tecnico.ulisboa.pt
ist189498	Maria Martins	https://git.rnl.tecnico.ulisboa.pt/ist189498	mailto:maria.d.martins@tecnico.ulisboa.pt

Q01

> When you run the command `terraform init` which plugins were installed? (You can copy the result of the command to insert in your report).

As we can see, terraform installed 4 providers: `google` , `kubernetes` , `helm` and `kubectl`



```
vagrant@mgmt:~/labs/k8scloudmesh$ terraform init
Initializing modules...
- gcp_gke in gcp_gke
- gcp_k8s in gcp_k8s

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/google from the dependency lock file
- Reusing previous version of hashicorp/kubernetes from the dependency lock file
- Reusing previous version of hashicorp/helm from the dependency lock file
- Reusing previous version of gavinbunney/kubectl from the dependency lock file
- Installing hashicorp/google v4.0.0...
- Installed hashicorp/google v4.0.0 (signed by HashiCorp)
- Installing hashicorp/kubernetes v2.6.1...
- Installed hashicorp/kubernetes v2.6.1 (signed by HashiCorp)
- Installing hashicorp/helm v2.3.0...
- Installed hashicorp/helm v2.3.0 (signed by HashiCorp)
- Installing gavinbunney/kubectl v1.13.0...
- Installed gavinbunney/kubectl v1.13.0 (self-signed, key ID AD64217B5ADD572F)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Q02

> Analyze briefly the `k8s-monitoring.tf` and interpret its purpose.

This file creates kubernetes resources for Prometheus and Grafana, according to the files present in the `gcp_k8s/monitoring` directory

Line no.	Interpretation
5-7, 19-21	Creates a data resource (one for Grafana, the other for Prometheus) that consists of a set of YAML manifests. These manifests were loaded from the path given in the `content` attribute and will be used to create the manifest resource in this same file
9-16, 23-30	Configures a remote kubernetes resource for each manifest previously created

Q03

> Analyze briefly the `k8s-istio.tf` and interpret its purpose.

This file declares the `ISTIO` resources that will implement the Service Mesh infrastructure

Line no.	Interpretation
5-16	Deploys the `ISTIO` Service Mesh, by using the pre-installed `ISTIO` charts
18-28	Deploys the `ISTIO` Discovery Service by using the pre-installed `ISTIO` charts

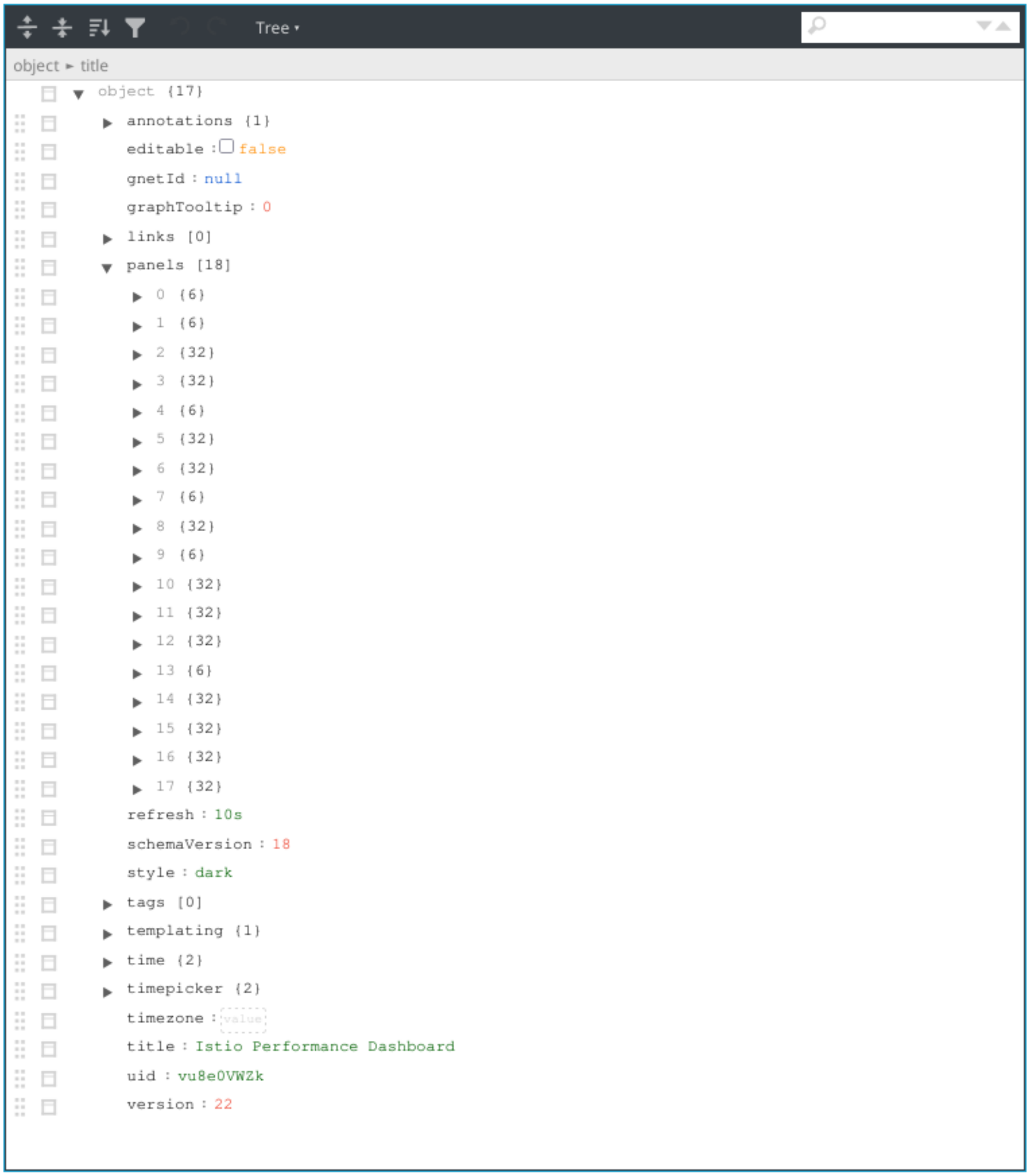
Q04

> In Module `gcp_k8s` there is a folder named monitoring containing two `.yaml` files. What is the purpose of those files?

These files configure Prometheus and Grafana Pods in the Kubernetes cluster and are loaded in `gcp_k8s/k8s-monitoring.tf` file.

The file `prometheus.yaml` also configures Prometheus Service, Pod and deployment and also its Role Based Access Control (RBAC).

The file `grafana.yaml` configures the Grafana Service and Pod, and also its data sources, the dashboard providers, and the dashboards it will present to the user. We can see huge JSON objects in this configuration file. After parsing them (see image below), we understood that each object configures a dashboard (we can note the `title` property) and has a set of `panels`, each corresponding to a respective metrics visualization.

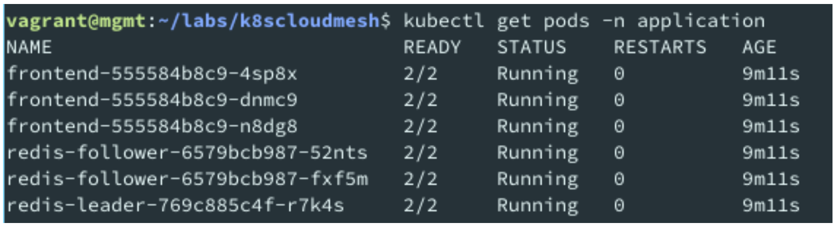


```
▼ object {17}
    ▶ annotations {1}
      editable : □ false
      gnetId : null
      graphTooltip : 0
    ▶ links [0]
    ▼ panels [18]
        ▶ 0 {6}
        ▶ 1 {6}
        ▶ 2 {32}
        ▶ 3 {32}
        ▶ 4 {6}
        ▶ 5 {32}
        ▶ 6 {32}
        ▶ 7 {6}
        ▶ 8 {32}
        ▶ 9 {6}
        ▶ 10 {32}
        ▶ 11 {32}
        ▶ 12 {32}
        ▶ 13 {6}
        ▶ 14 {32}
        ▶ 15 {32}
        ▶ 16 {32}
        ▶ 17 {32}
      refresh : 10s
      schemaVersion : 18
      style : dark
    ▶ tags [0]
    ▶ templating {1}
    ▶ time {2}
    ▶ timepicker {2}
      timezone : value
      title : Istio Performance Dashboard
      uid : vu8e0VWZk
      version : 22
```

Q05

> Use the command `kubectl get pods` to get the information about the application Pods and report the result. How many containers were reported in each Pod?

After running the command `kubectl get pods -n application`, we got the following output:



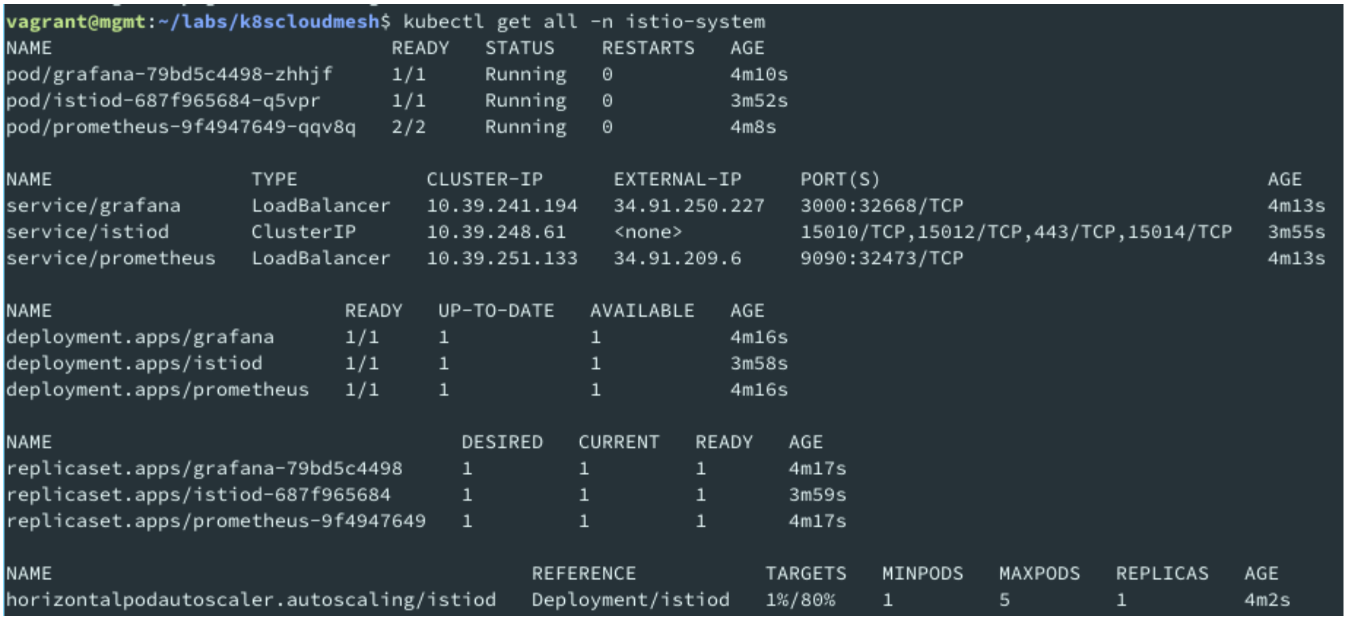
```
vagrant@mgmt:~/labs/k8scloudmesh$ kubectl get pods -n application
NAME                              READY   STATUS    RESTARTS   AGE
frontend-555584b8c9-4sp8x         2/2     Running   0          9m11s
frontend-555584b8c9-dnmc9         2/2     Running   0          9m11s
frontend-555584b8c9-n8dg8         2/2     Running   0          9m11s
redis-follower-6579bcb987-52nts   2/2     Running   0          9m11s
redis-follower-6579bcb987-fxf5m   2/2     Running   0          9m11s
redis-leader-769c885c4f-r7k4s     2/2     Running   0          9m11s
```

This shows that there are 2 containers running in each Pod. This is due to the Service Mesh architecture: One container is the service instance and handles the application logic. The other one is the proxy instance (a.k.a. sidecar), which handles interservice communication and "abstracts away security and connection concerns from the development of the application" (src).

Q06

> Check all the information in the istio-system and report the result. Which Pods in state running were reported. Are there replica sets reported?

After running the command `kubectl get all -n istio-system`, we got the following output:



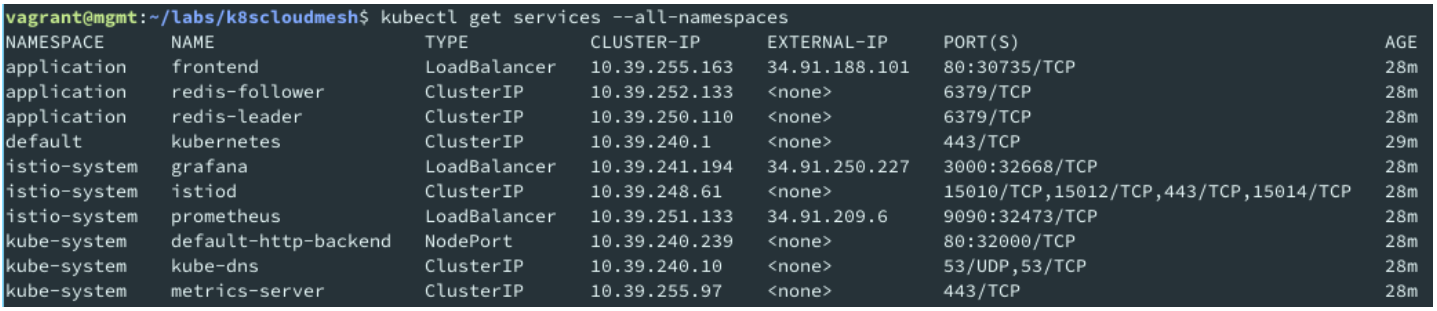
```
vagrant@mgmt:~/labs/k8scloudmesh$ kubectl get all -n istio-system
NAME                              READY   STATUS    RESTARTS   AGE
pod/grafana-79bd5c4498-zhhjf      1/1     Running   0          4m10s
pod/istiod-687f965684-q5vpr       1/1     Running   0          3m52s
pod/prometheus-9f4947649-qqv8q    2/2     Running   0          4m8s

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)                                 AGE
service/grafana      LoadBalancer   10.39.241.194   34.91.250.227   3000:32668/TCP                          4m13s
service/istiod       ClusterIP      10.39.248.61    <none>          15010/TCP,15012/TCP,443/TCP,15014/TCP   3m55s
service/prometheus   LoadBalancer   10.39.251.133   34.91.209.6     9090:32473/TCP                          4m13s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/grafana      1/1     1            1           4m16s
deployment.apps/istiod       1/1     1            1           3m58s
deployment.apps/prometheus   1/1     1            1           4m16s

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/grafana-79bd5c4498      1         1         1       4m17s
replicaset.apps/istiod-687f965684       1         1         1       3m59s
replicaset.apps/prometheus-9f4947649    1         1         1       4m17s

NAME                                         REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/istiod   Deployment/istiod   1%/80%    1         5         1          4m2s
```

We can see that in the `istio-system` namespace, there are three Pods running: one for `Prometheus`, another for `Grafana` and a last one for `istiod` (`ISTIO` discovery service) and each Pod has a service assigned to it. The command reported three `replicasets`, one for each Pod. Each contains one replica running, as desired.

Q07

> Get the information about all the services running in the Kubernetes cluster, and write it in the report.

After running the command `kubectl get services --all-namespaces`, we were able to get information about all the running Services of this cluster, as shown below:



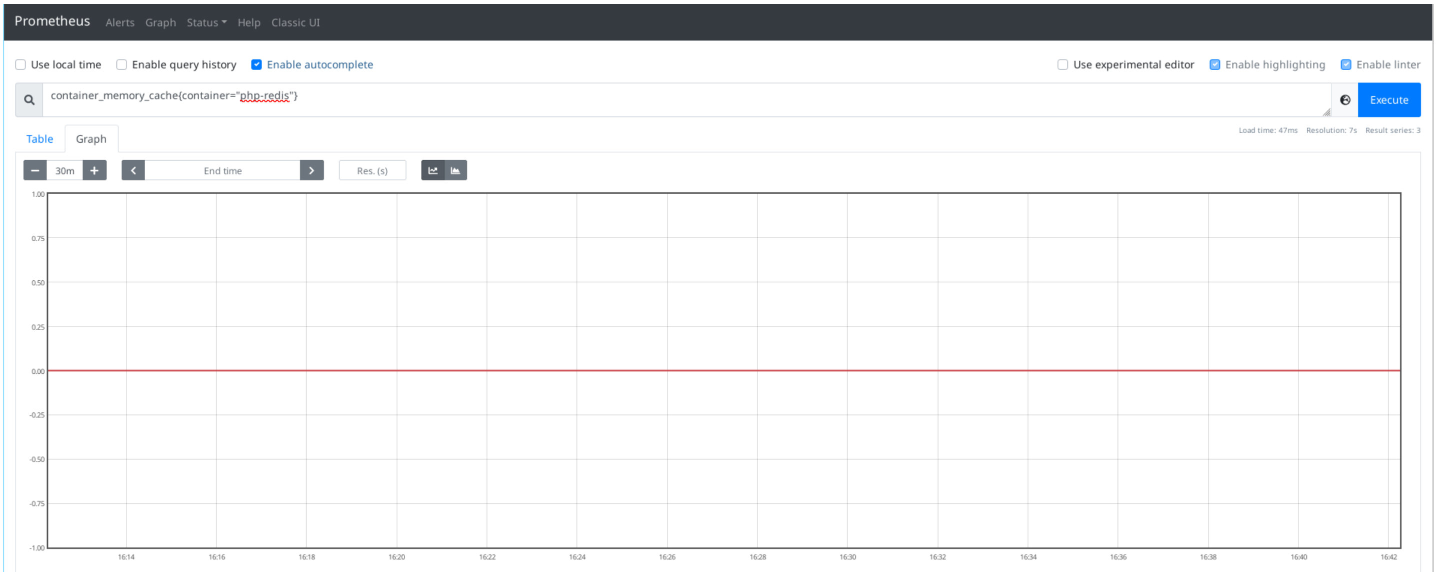
```
vagrant@mgmt:~/labs/k8scloudmesh$ kubectl get services --all-namespaces
NAMESPACE      NAME                   TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)                                     AGE
application    frontend               LoadBalancer   10.39.255.163   34.91.188.101   80:30735/TCP                                28m
application    redis-follower         ClusterIP      10.39.252.133   <none>          6379/TCP                                    28m
application    redis-leader           ClusterIP      10.39.250.110   <none>          6379/TCP                                    28m
default        kubernetes             ClusterIP      10.39.240.1     <none>          443/TCP                                     29m
istio-system   grafana                LoadBalancer   10.39.241.194   34.91.250.227   3000:32668/TCP                              28m
istio-system   istiod                 ClusterIP      10.39.248.61    <none>          15010/TCP,15012/TCP,443/TCP,15014/TCP       28m
istio-system   prometheus             LoadBalancer   10.39.251.133   34.91.209.6     9090:32473/TCP                              28m
kube-system    default-http-backend   NodePort       10.39.240.239   <none>          80:32000/TCP                                28m
kube-system    kube-dns               ClusterIP      10.39.240.10    <none>          53/UDP,53/TCP                               28m
kube-system    metrics-server         ClusterIP      10.39.255.97    <none>          443/TCP                                     28m
```

We can see several services in this output, belonging to one of three user-defined namespaces (`application` , `istio-system` and `kube-system`). We can see that only services directly accessible from the outside of the cluster are `grafana` , `prometheus` and `frontend` , each having its own external IP. We can note that despite some services correspond to replicated Pods (e.g. `redis-follower`), each service is listed only once, indicating that the same service is used by every associated replica. This demonstrates the transparency of replication in Kubernetes.

Q08

> Which metrics (and targets) have you selected to observe in Prometheus? Post a screenshot of the Prometheus Dashboard showing the values of some of the metrics for kubernetes-pods.

Prometheus allows a user to select any of the available metrics and displays a graph with the targeted data from the selected labels. Below we can see the `container_memory_cache` metric from `php-redis` containers. This graph shows a flat line since the memory cache usage was constant for the entire process.



□ Use local time □ Enable query history ☑ Enable autocomplete □ Use experimental editor ☑ Enable highlighting ☑ Enable linter

container_memory_cache{container="php-redis"}

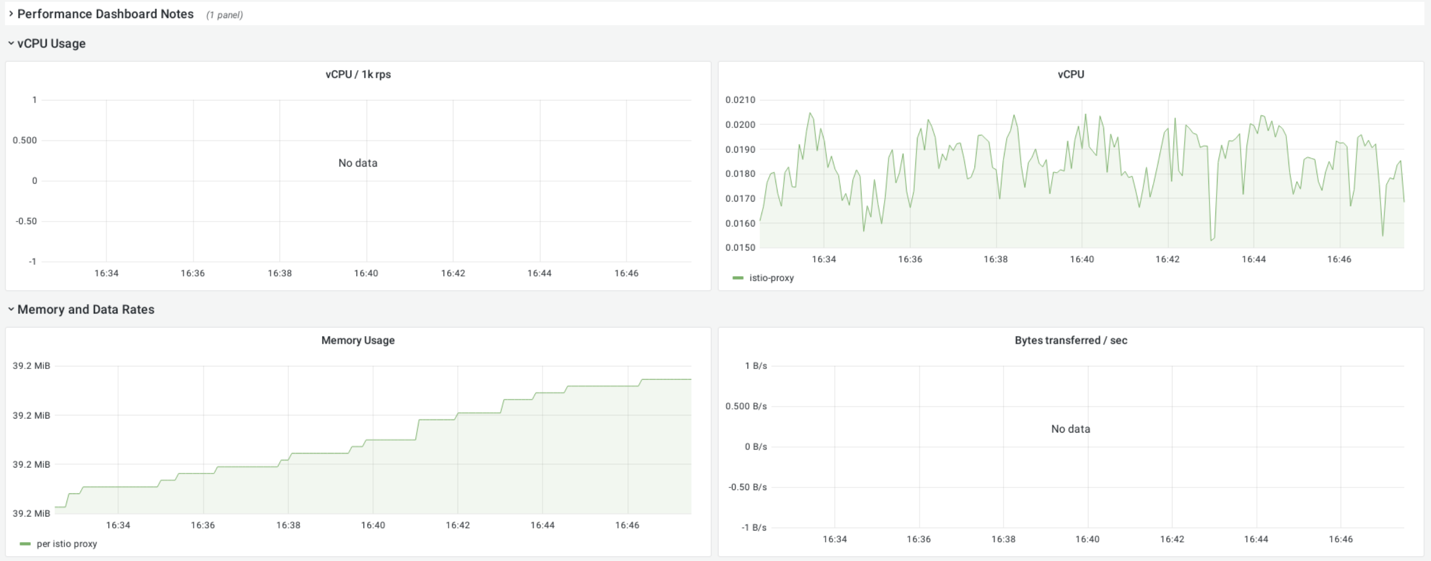
Load time: 47ms Resolution: 7s Result series: 3

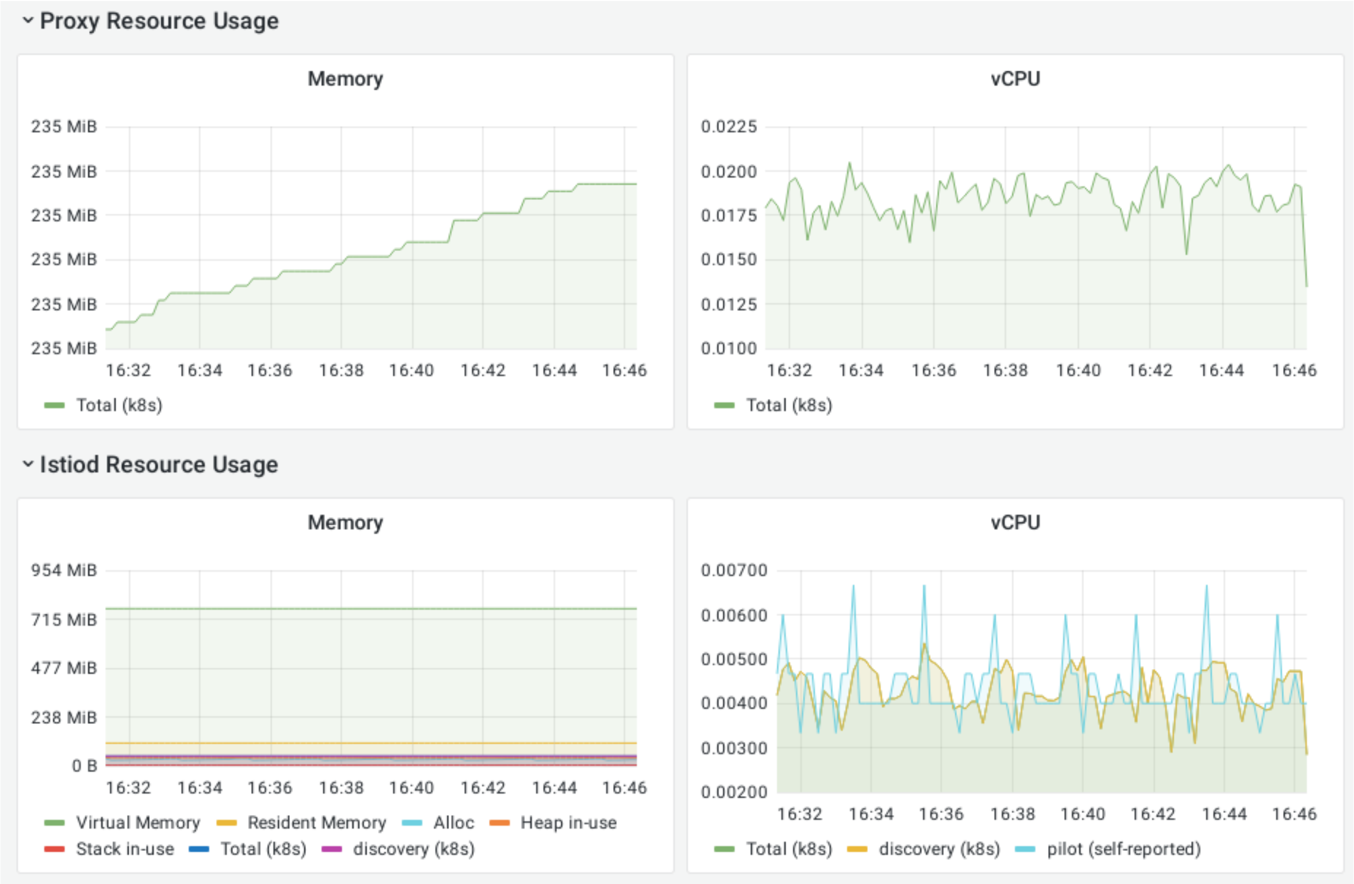
Q09

> Select a Grafana Performance Dashboard. Which Metrics compose the Dashboard? Post a screenshot of that Dashboard in your report.

The charts on this dashboard show Istio main components cost in terms of resource utilization under steady load, namely the vCPU usage, Memory footprint of the components, the amount of bytes flowing through each component and the number of Goroutines.

Below we can see some of these graphs:



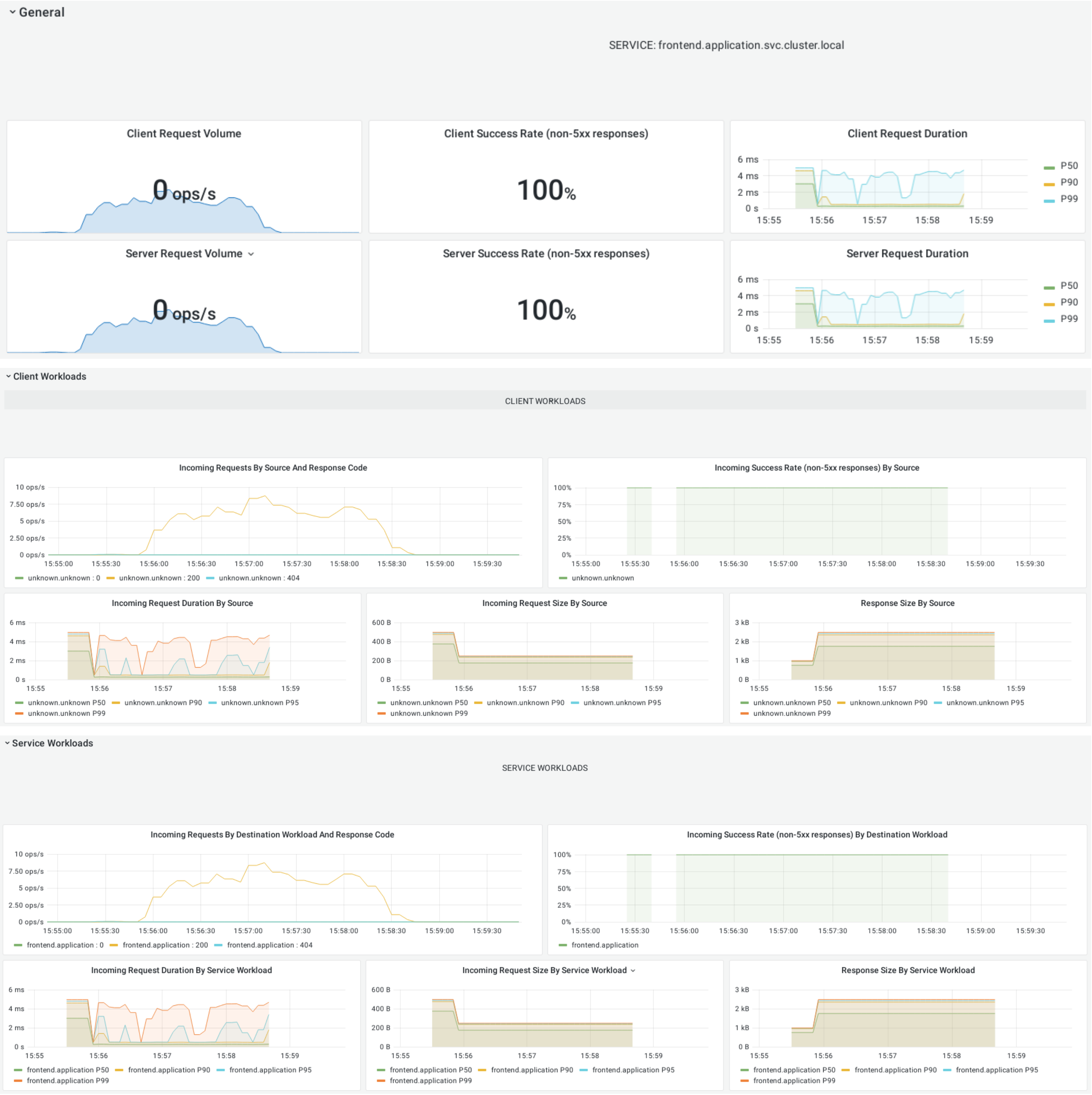


Q10

> Select a Grafana Service Dashboard. Which Metrics compose the Dashboard? Post a screenshot of that Dashboard in your report.

The charts on this dashboard show metrics on the running services, such as the number of requests, the percentage of successful requests, the client and server request durations, the number of sent and received bytes, request duration and size, response size and duration

Below we can see some of these graphs:



General

SERVICE: frontend.application.svc.cluster.local

Client Request Volume: 0 ops/s

Client Success Rate (non-5xx responses): 100%

Client Request Duration

Server Request Volume: 0 ops/s

Server Success Rate (non-5xx responses): 100%

Server Request Duration

Client Workloads

CLIENT WORKLOADS

Incoming Requests By Source And Response Code

Incoming Success Rate (non-5xx responses) By Source

Incoming Request Duration By Source

Incoming Request Size By Source

Response Size By Source

Service Workloads

SERVICE WORKLOADS

Incoming Requests By Destination Workload And Response Code

Incoming Success Rate (non-5xx responses) By Destination Workload

Incoming Request Duration By Service Workload

Incoming Request Size By Service Workload

Response Size By Service Workload

```
 -----------
< TEAM 20 A >
 -----------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/
                ||----w
                ||     |
```