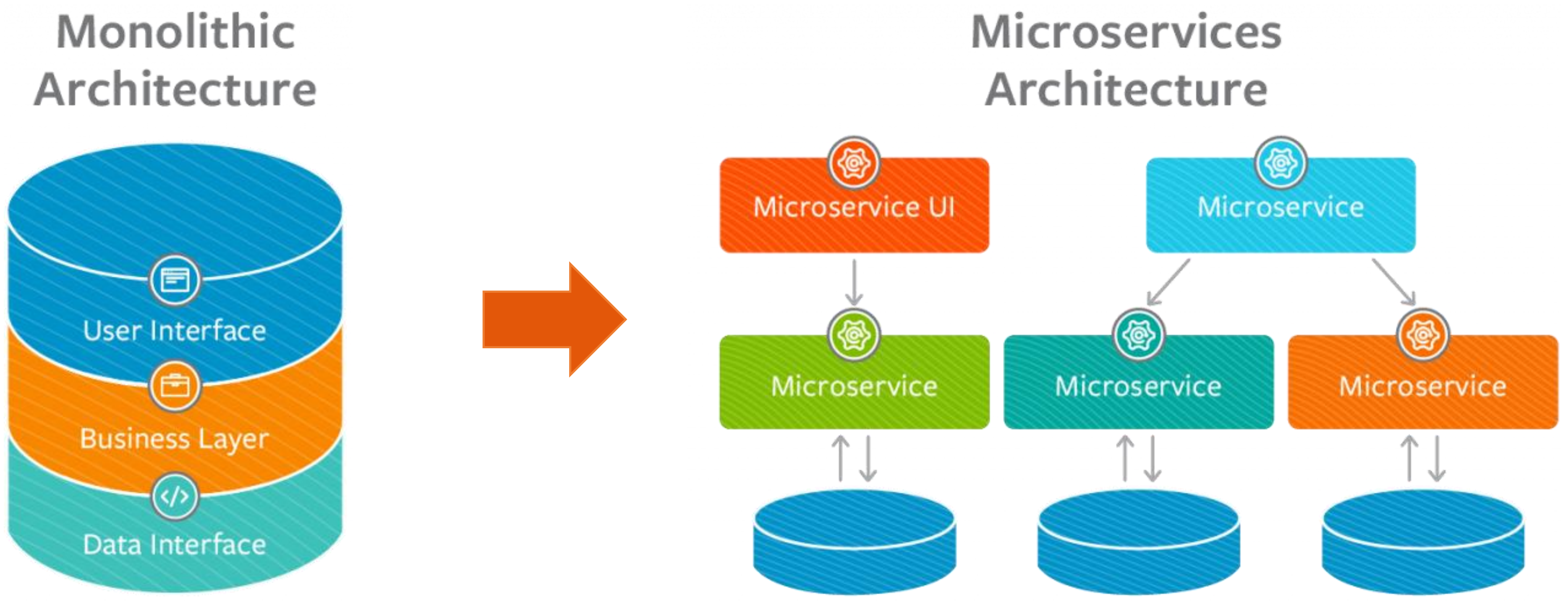
TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Kubernetes

Sistemi Distribuiti e Cloud Computing A.A. 2020/21
Laurea Magistrale Ingegneria Informatica

Fabiana Rossi
f.rossi@ing.uniroma2.it

FROM MONOLITHIC TO MICROSERVICES



- **What is container orchestration?**

  - **Container orchestration** is the automation of much of the operational effort required to run containerized workloads and services.
    - This includes a wide range of things software teams need to manage a container's lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing and more.

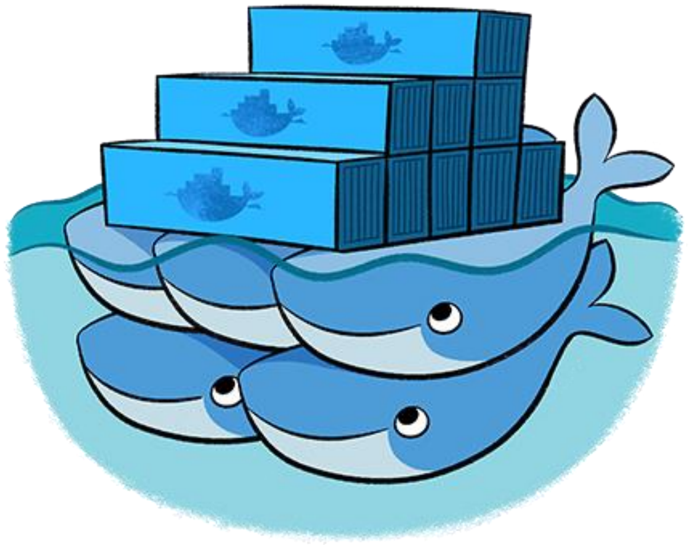
- **Container orchestration versus Docker**

  - Docker is a specific platform for building containers, including the Docker Engine container runtime, whereas container orchestration is a broader term referring to automation of any container's lifecycle. Docker also includes Docker Swarm, which is the platform's own container orchestration tool that can automatically start Docker containers.

ORCHESTRATION FRAMEWORKS

kubernetes*

Apache MESOS™



* Jawarneh, I.M.A., Bellavista, P., Bosi, F., Foschini, L., Martuscelli, G., Montanari, R., Palopoli, A.: *Container orchestration engines: A thorough functional and performance comparison.* In Proc. of IEEE ICC 2019. pp. 1–6 (2019)

What Kubernetes is

- **Kubernetes** is an **open source** container orchestration engine for automating deployment, scaling, and management of containerized application.
- Originally an open source project launched by Google and now part of the Cloud Native Computing Foundation (CNCF).
- Kubernetes is **highly extensible** and **portable**
  - it can run in a wide range of environments and be used in conjunction with other technologies, such as service meshes.
- Kubernetes is considered **highly declarative**
  - Developers and administrators use it to describe how they want a system to behave, and then Kubernetes executes that desired state in dynamic fashion.

What Kubernetes is

Kubernetes provides you with:

- **Service discovery and load balancing:** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration:** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- **Automated rollouts and rollbacks:** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate.

What Kubernetes is

kubernetes

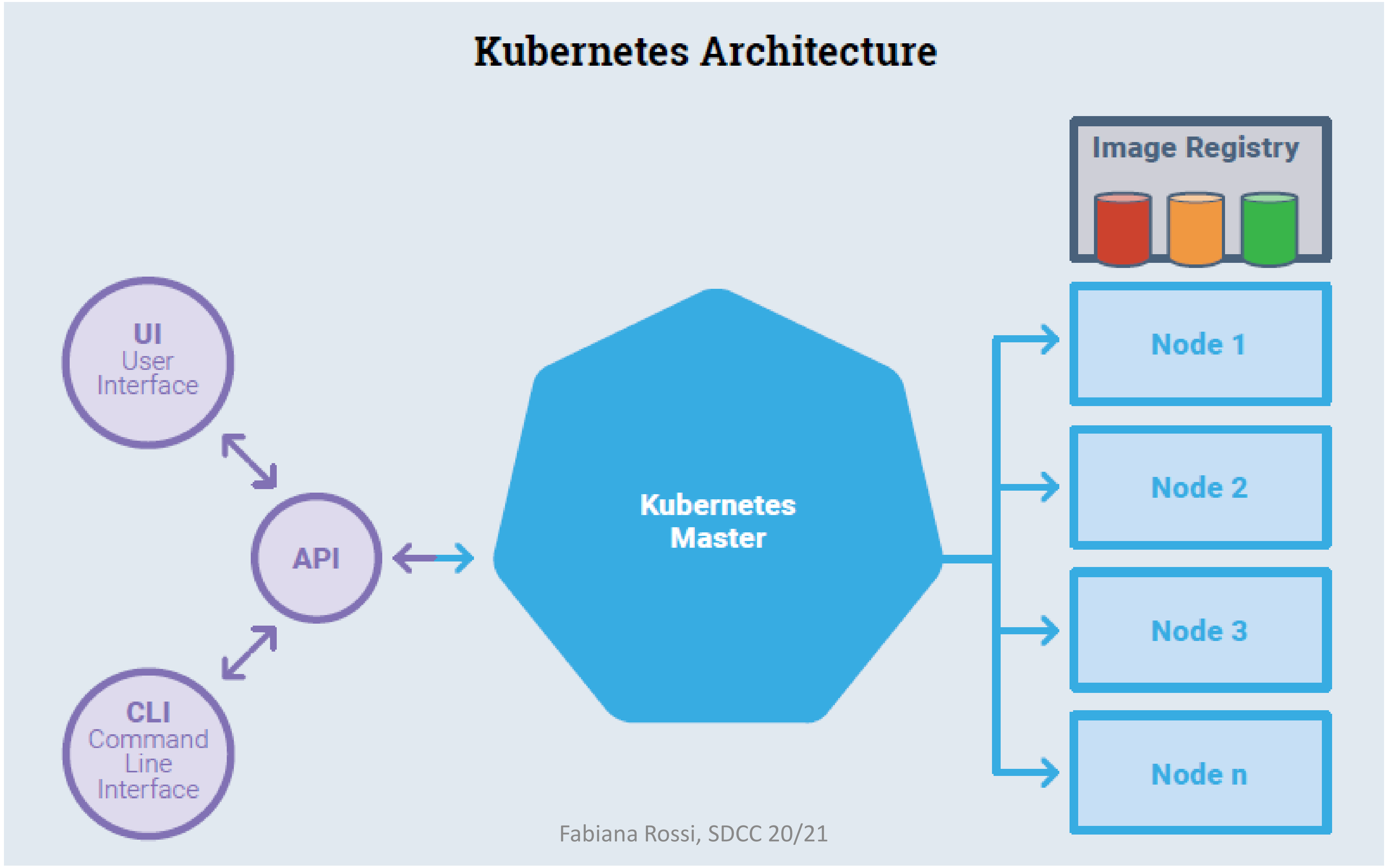
- **Scheduling:** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks.
- **Self-healing:** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Secret and configuration management:** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

What Kubernetes is not

Kubernetes:

- Does not limit the types of applications supported.
    - Kubernetes aims to support an extremely diverse variety of workloads, including stateless, stateful, and data-processing workloads. If an application can run in a container, it can run on Kubernetes.
- Does not deploy source code and does not build your application.
- Does not provide application-level services, such as middleware (e.g., message buses), data-processing frameworks (e.g., Spark), databases (e.g., MySQL), caches, nor cluster storage systems (e.g., Ceph) as built-in services.
- Does not dictate logging, monitoring, or alerting solutions.
    - It provides some integrations as proof of concept, and mechanisms to collect and export metrics.

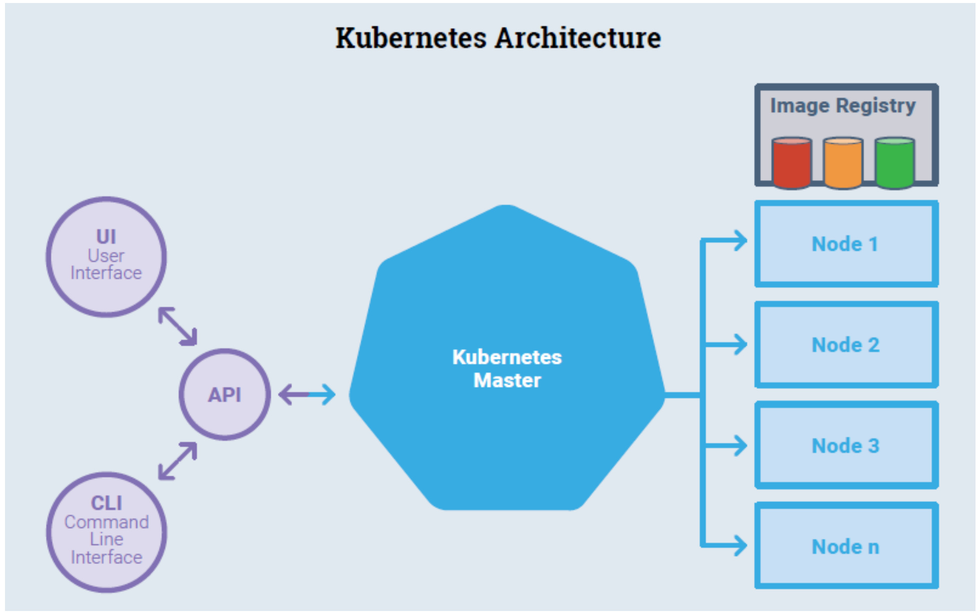
Kubernetes Architecture



Kubernetes Components

- When you deploy Kubernetes, you get a cluster.
- A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications.
  - Every cluster has at least one worker node;
  - The worker node(s) host the Pods that are the components of the applications.
- The master manages the worker nodes and the Pods in the cluster

Kubernetes Architecture

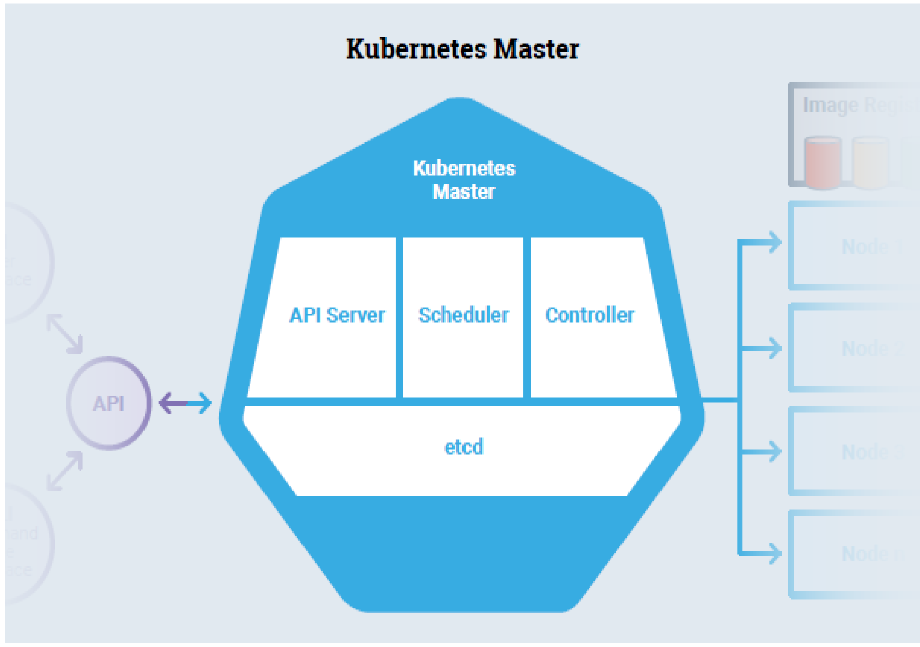


- The master is responsible for:
  - exposing the Kubernetes (REST) API,
  - scheduling the applications,
  - managing the cluster,
  - directing communications across the entire system,
  - monitoring the containers running in each node as well as the health of all the registered nodes.
- The nodes that are responsible for scheduling and running the containerized applications
  - Container images, which act as the deployable artifacts, must be available to the Kubernetes cluster through a private or public image registry.

Kubernetes Master

The Kubernetes master runs the following components that form the control plane:

- **API server**: the front-end for the Kubernetes control plane that exposes the Kubernetes API
  - kube-apiserver
  - designed to scale horizontally
- **etcd**: is a persistent, lightweight, distributed key-value data store that maintains the entire state of the cluster at any given point of time
- **scheduler**: watches for newly created Pods with no assigned node, and selects a node for them to run on
  - kube-scheduler
- **controller**: control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current cluster state to the desired cluster state



Kubernetes Nodes

- Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

- Kubernetes Nodes components:
  - *kubelet*: agent that makes sure that containers are running in a Pod.
  - *kube-proxy*: implemented as a network proxy and a load balancer.
    - It routes traffic to the appropriate container based on its IP address and the port number of the incoming request.
    - Part of the Kubernetes Service concept.
  - *container runtime*: the software that is responsible for running containers.
  - Kubernetes supports several container runtimes:
    - Docker,
    - containerd,
    - CRI-O,
    - any implementation of the Kubernetes CRI (Container Runtime Interface).

Kubernetes Object

- Kubernetes objects are persistent entities provided by Kubernetes for deploying, maintaining, and scaling applications.

- Kubernetes uses these entities to represent the state of your cluster.

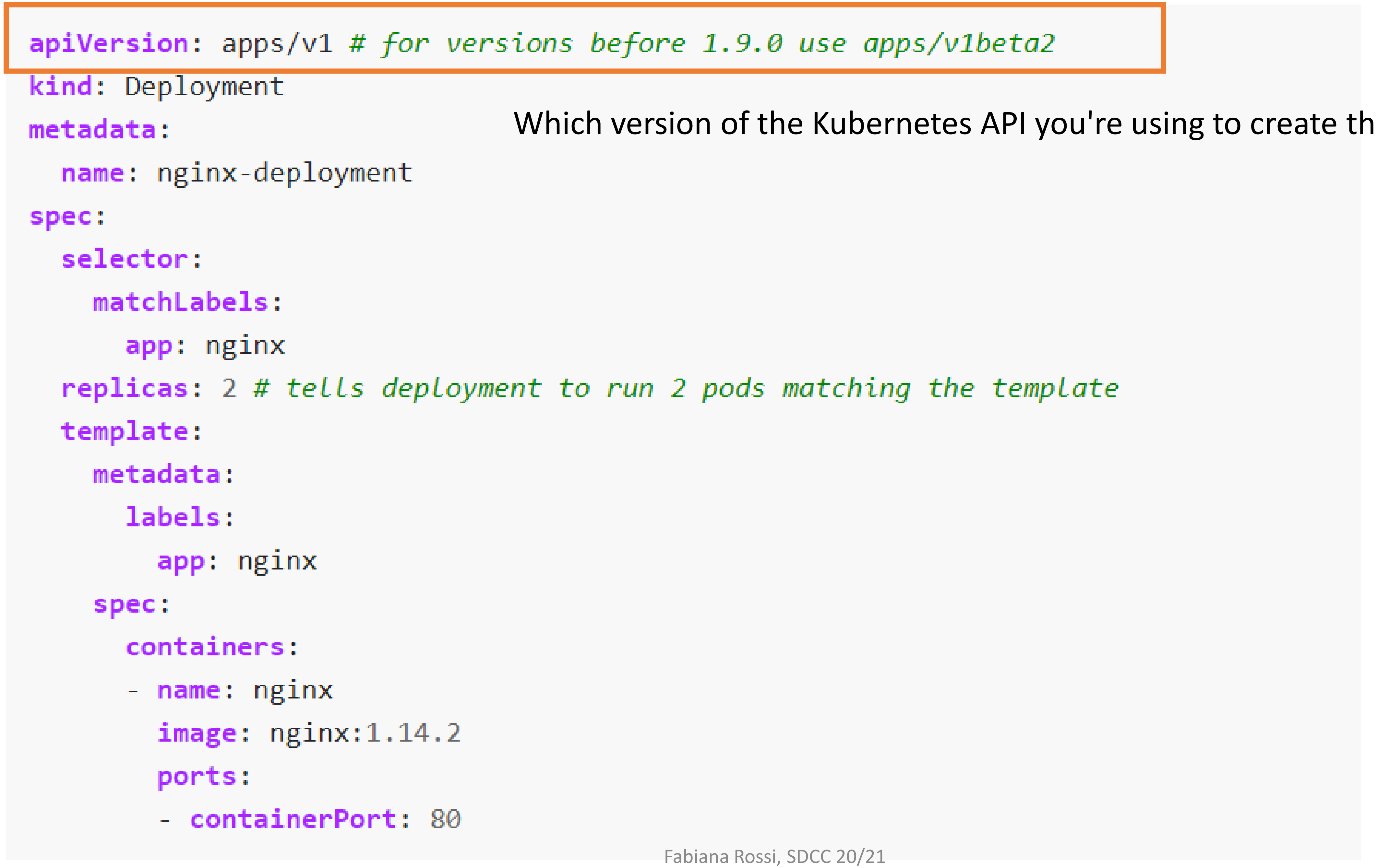
- The objects can describe:
  - What containerized applications are running (and on which nodes)
  - The resources available to those applications
  - The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance

To work with Kubernetes objects (i.e., to create, modify, or delete them) you'll need to use the Kubernetes API.
When you use the kubectl command-line interface, for example, the CLI makes the necessary Kubernetes API calls for you.

Label and Selector

- Any object in Kubernetes may have **key-value** pairs associated with it — additional metadata for identifying and grouping objects sharing a common attribute or property.
- Kubernetes refers to these key-value pairs as *labels*.
- Labels do not provide uniqueness
  - In general, we expect many objects to carry the same label(s).
- Via a *label selector*, the client/user can identify a set of objects.
- The label selector is the core grouping primitive in Kubernetes.

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

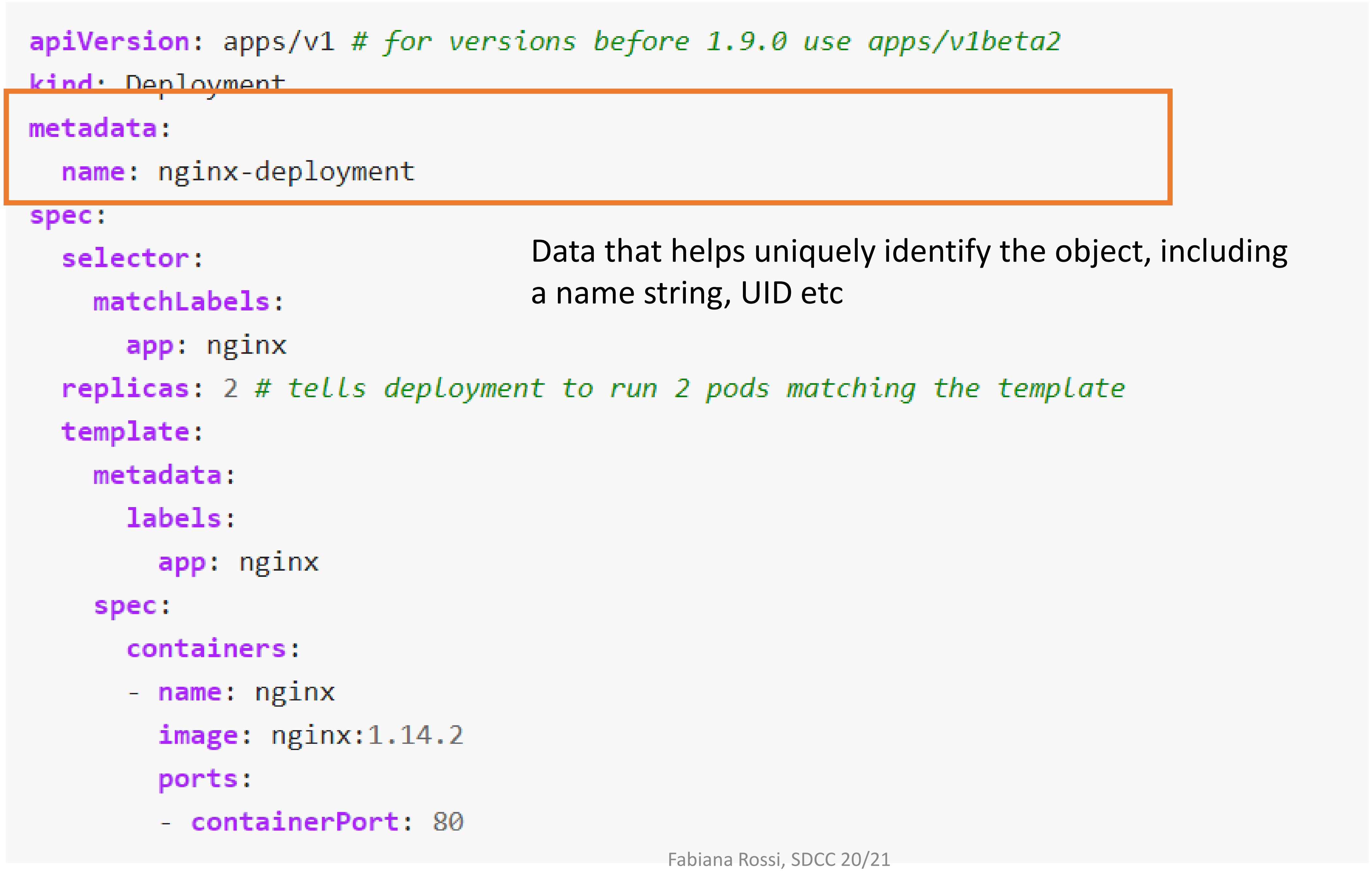


```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Which version of the Kubernetes API you're using to create this object

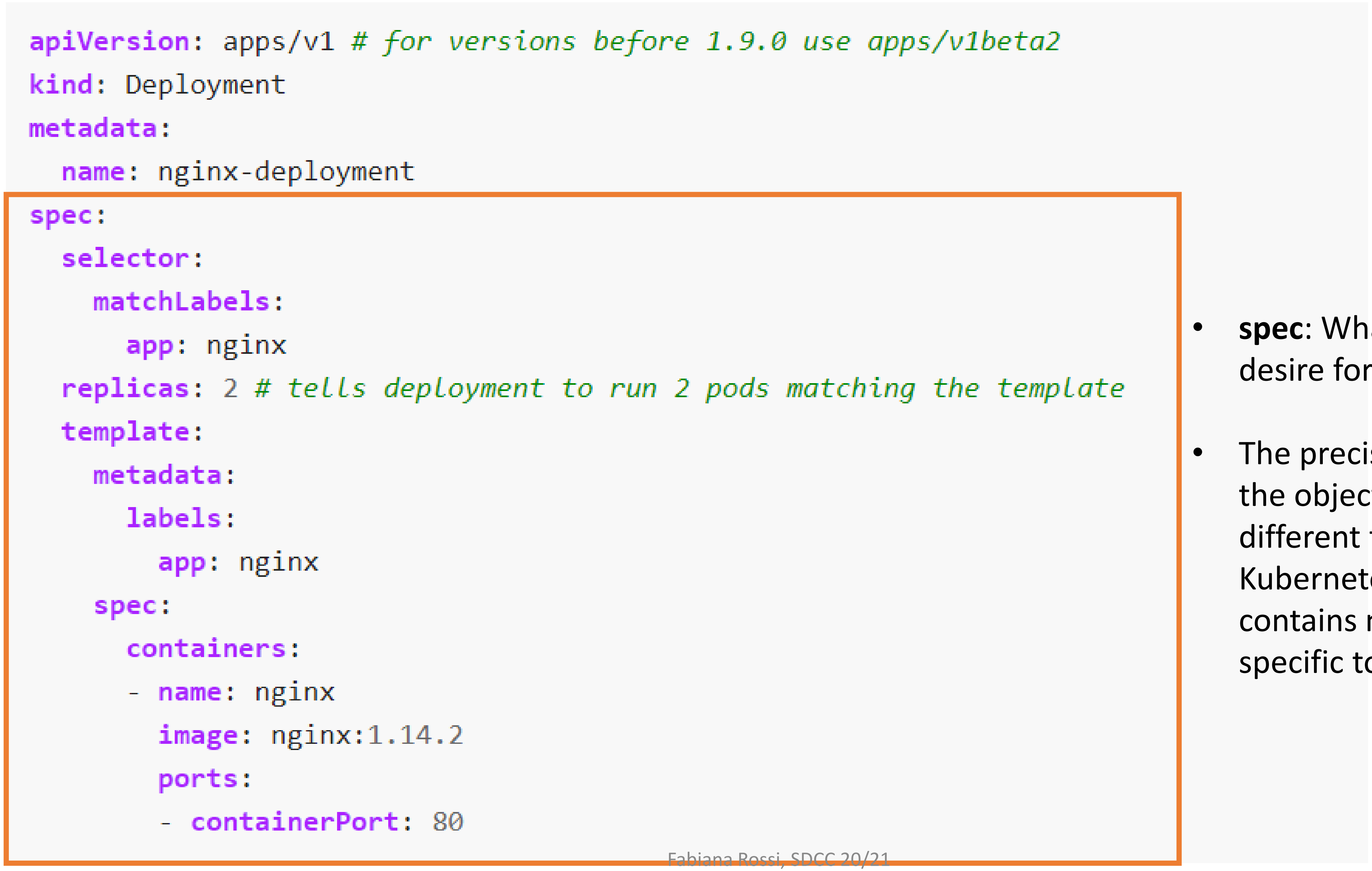
```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

What kind of object you want to create



```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Data that helps uniquely identify the object, including a name string, UID etc



```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

- **spec**: What state you desire for the object.
- The precise format of the object spec is different for every Kubernetes object, and contains nested fields specific to that object

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

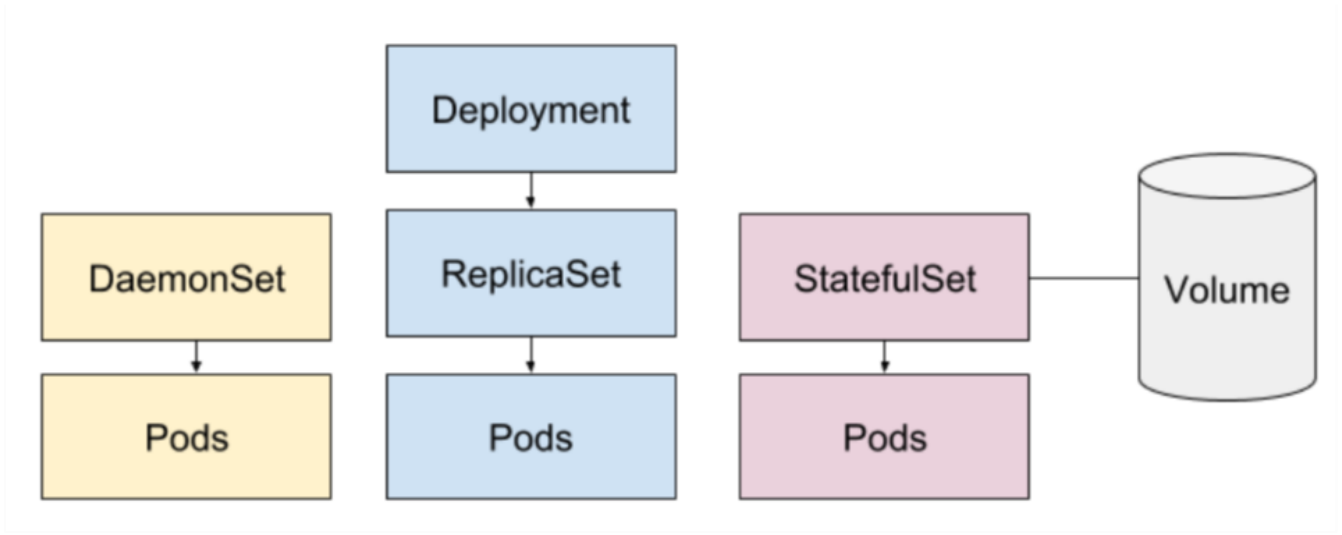
Pod label «app: nginx»

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

The Deployment selects Pods with the label «app:nginx»

Basic Kubernetes Objects

- Pod
- Deployment, ReplicaSet
- DaemonSet
- StatefulSet
- Service
- Secret



Pods

- *Pods* are the smallest deployable units of computing that you can create and manage in Kubernetes.
- A *Pod* is a group of one or more containers, with shared storage/network resources, and a specification for how to run the containers.
- A Pod's contents are always co-located and co-scheduled.

Pod Management

- To create and manage multiple Pods, Kubernetes defines multiple resource types:
    - Deployment
    - StatefulSet
    - DaemonSet

Deployment

- **Deployment**: represents a set of multiple, identical Pods with no unique identities.
  - It runs multiple replicas Pods and automatically replaces any instances that fail or become unresponsive.
  - In this way, Deployments ensure that one or more instances of Pods are available to serve user requests.

- To create the Deployment: kubectl apply –f nginx-deployment.yaml
- Run kubectl get deployments to check if the Deployment was created

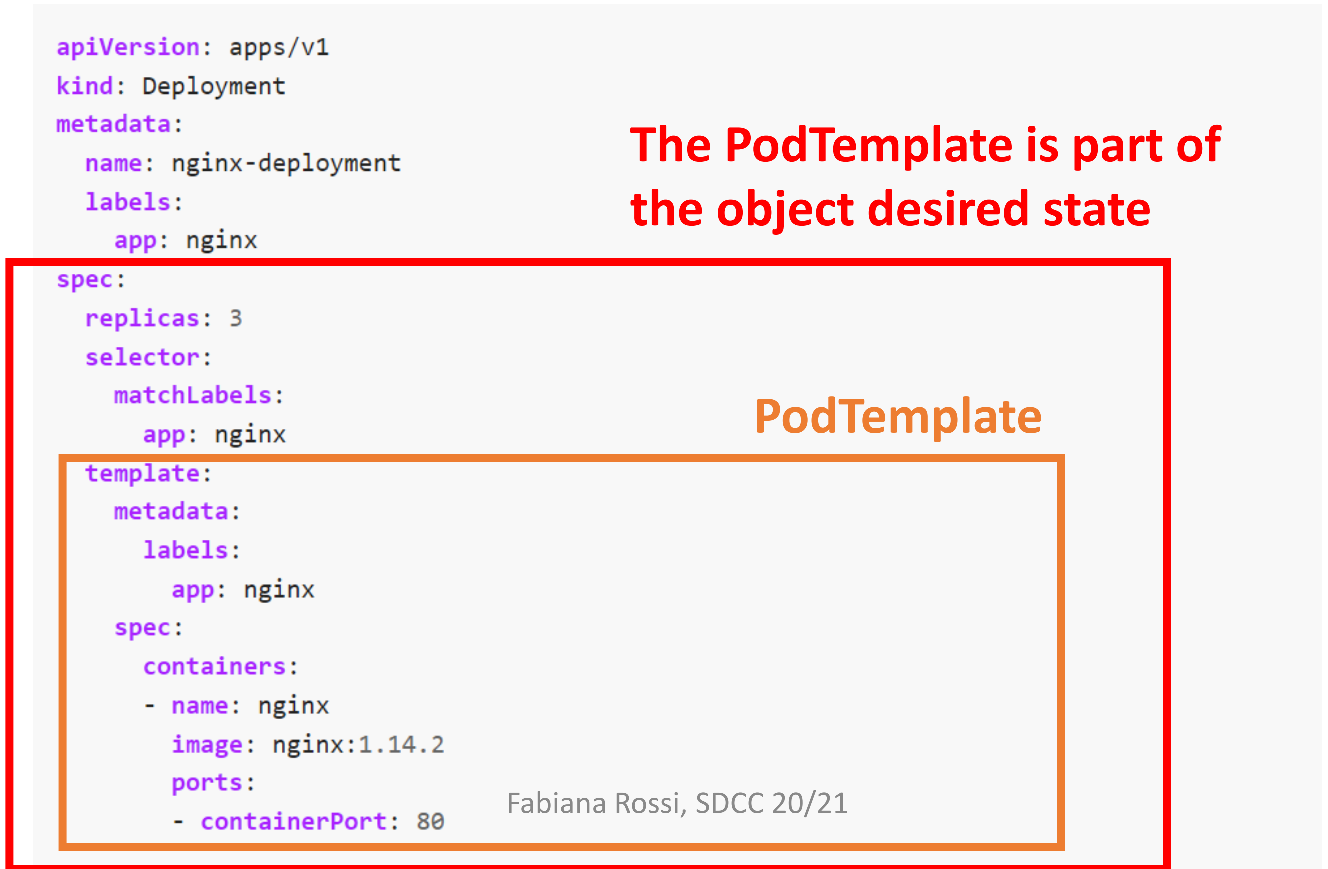
```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3 .          3           18s
```

- To see the pods: kubectl get pods

```
NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-75675f5897-7ci7o   1/1       Running   0          18s
nginx-deployment-75675f5897-kzszj   1/1       Running   0          18s
nginx-deployment-75675f5897-qqcnn   1/1       Running   0          18s
```

PodTemplate

- PodTemplates are specifications for creating Pods, and are included in resource objects such as Deployment object.

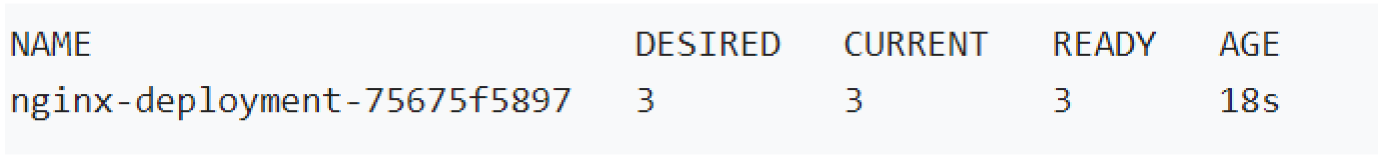


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

ReplicaSet

- A **ReplicaSet** is the next-generation of ReplicationControllers
- It ensures that a specified number of pod replicas are running at any given time.
  - It fulfills its purpose by creating and deleting Pods as needed to reach the desired number. When a ReplicaSet needs to create new Pods, it uses its Pod template.

- To see the ReplicaSet run kubectl get rs



```
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-75675f5897   3         3         3       18s
```

- A Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods

Pod Management

- **StatefulSet**: manages deployment and scaling of a set of Pods, with durable storage and persistent identifiers for each pod.
    - Unlike a Deployment, a StatefulSet maintains a sticky identity for each of its Pods.

- **DaemonSet**: ensures that all (or some) nodes run a copy of a Pod.
    - As nodes are added to the cluster, Pods are added to them.
    - As nodes are removed from the cluster, those Pods are garbage collected.
    - Deleting a DaemonSet will clean up the Pods it created.

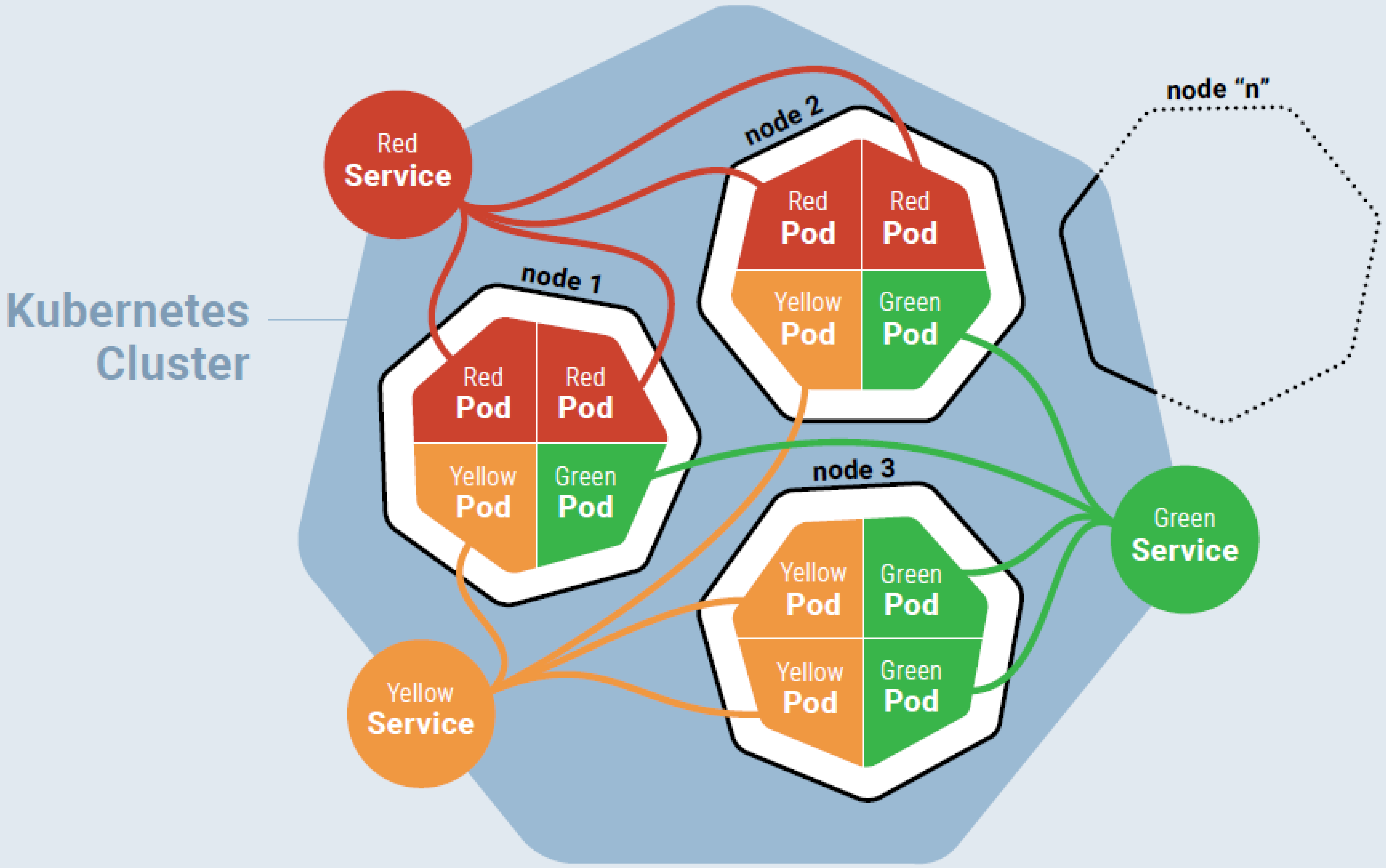
Controller

- In Kubernetes, **controllers** are **control loops** that watch the state of the cluster, then make or request changes where needed.
- A controller tracks at least one Kubernetes resource object.
- The controller(s) for that resource are responsible for making the current state come closer to that desired state (specified in the *spec* field).
- Kubernetes comes with a set of controllers that run inside the *kube-controller-manager*
- The Deployment controller is an example of controller that come as part of Kubernetes itself ("built-in" controllers).

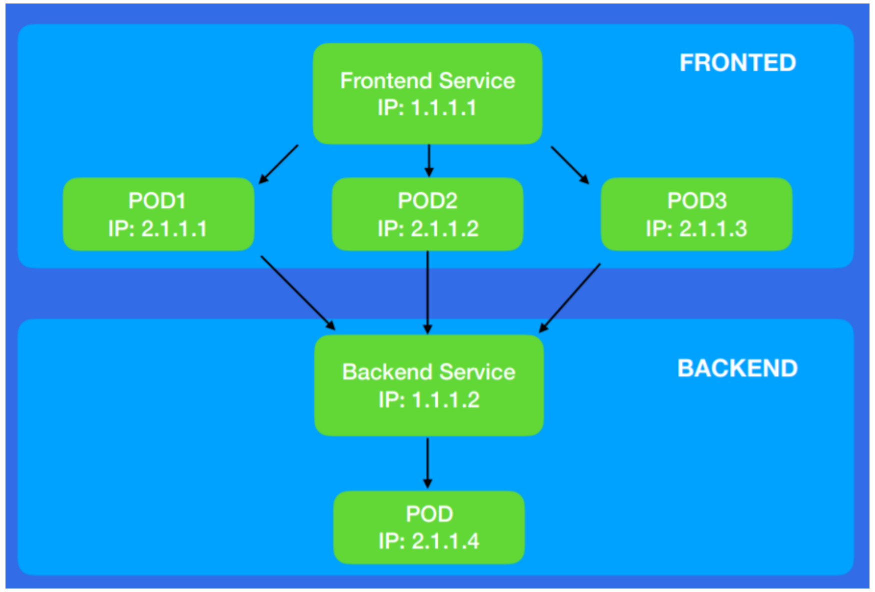
Service

- The services model in Kubernetes relies upon the most basic, though most important, aspect of services: *discovery*.
- In Kubernetes, a **Service** is an abstraction which defines a logical set of Pods and a policy by which to access them.
- The set of Pods targeted by a Service is usually determined by a selector.
- Services ensure that traffic is always routed to the appropriate Pod within the cluster, regardless of the node on which it is scheduled.
- Each service exposes an **IP address**, and may also expose a **DNS endpoint**, both of which will never change.
    - Internal or external consumers that need to communicate with a set of pods will use the service’s IP address, or its more generally known DNS endpoint.

How Services in a Cluster Map to Functions in Pods



Service



- **Services** may be configured to expose pods to **internal** and **external consumers**.

- Different Service Types:
  - **ClusterIP**: Exposes the Service on a cluster-internal IP.
    - Choosing this value makes the Service only reachable from within the cluster. This is the default ServiceType.

  - **NodePort**: Exposes the Service on each Node's IP at a static port (the NodePort).
    - A ClusterIP Service, to which the NodePort Service routes, is automatically created.
    - To contact the NodePort Service from outside the cluster: <NodeIP>:<NodePort>.

Service

```yaml
kind: Service
apiVersion: v1
metadata:
  name: port-example-svc
spec:
  # Make the service externally visible via the node
  type: NodePort

  ports:
    # Which port on the node is the service available through?
    - nodePort: 31234

    # Inside the cluster, what port does the service expose?
    - port: 8080

    # Which port do pods selected by this service expose?
    - targetPort:

  selector:
    # ...
```

- **nodePort**: This setting makes the service visible outside the Kubernetes cluster by the node's IP address and the port number declared in this property.
  - The service also has to be of type NodePort.
- **port**: Expose the service on the specified port internally within the cluster. That is, the service becomes visible on this port, and will send requests made to this port to the pods selected by the service.
- **targetPort**: This is the port on the pod that the request gets sent to. Your application needs to be listening for network requests on this port for the service to work.

Secret

- **Secrets** are secure objects which store sensitive data, such as passwords, OAuth tokens, and SSH keys, in your clusters.
- Storing sensitive data in Secrets is more secure than plaintext in Pod specifications.
- Using Secrets gives you control over how sensitive data is used, and reduces the risk of exposing the data to unauthorized users.

Kube-scheduler

- Kubernetes includes a default scheduler, named *kube-scheduler*, which allocates pods on worker nodes according to an extensible policy.
- The default scheduling strategy is spread
- For each pod, Kube-scheduler chooses a destination node through two-steps:
  - **filtering step**
  - **Scoring step**

Kube-scheduler

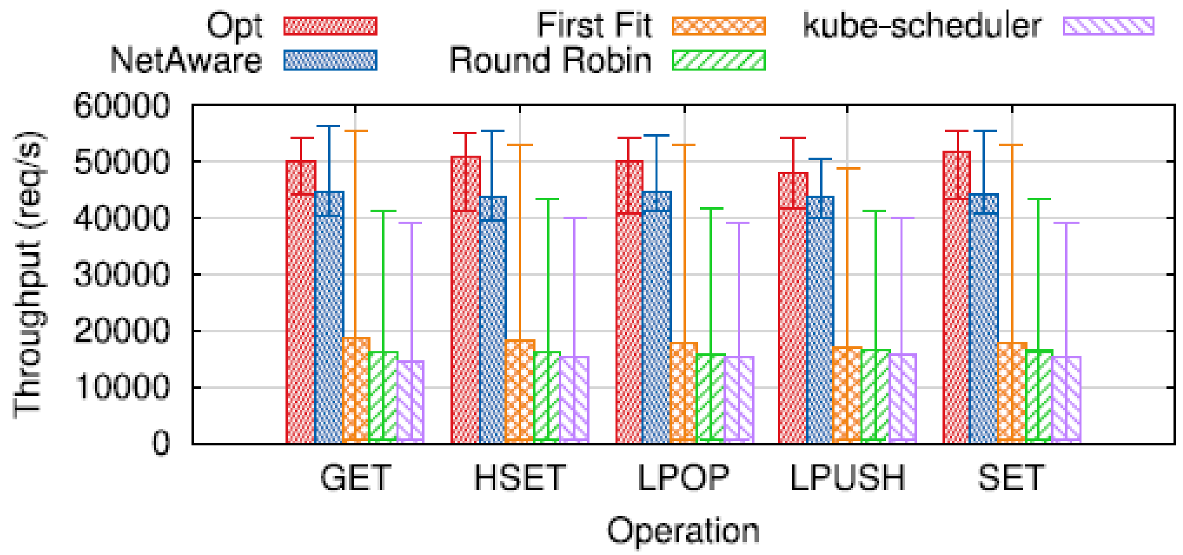
- **Filtering step**: Kube-scheduler identifies those worker nodes that can run the pod by applying a set of filters (i.e., **predicates**).
    - The first step discards the nodes that cannot satisfy the required resources and the label matching defined in the pod configuration file.

- **Scoring step**: the Kube-scheduler ranks the remaining nodes through a set of **priority functions**.
    - Each priority function assigns a score to each node and the final score of each node is calculated by adding up all the given weighted scores.
    - The one having the highest score is chosen to run the pod.
    - If multiple nodes achieve the same score, one of them is randomly selected.

Kubernetes and a Geo-distributed Environment

Case study*:

- Deployment of a Redis Cluster (3 master nodes)
  - Redis is a popular key–value data store often included in web applications to implement in-memory distributed caching.
- Computing infrastructure: 12 VMs in 4 Cloud regions



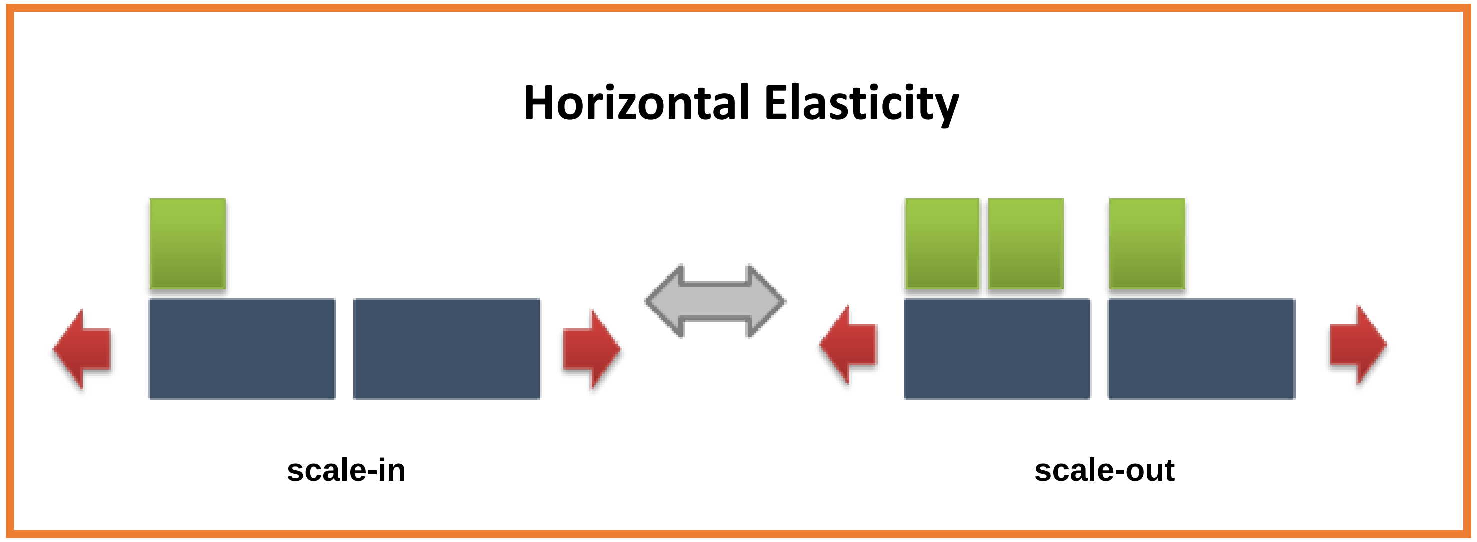
*** Source**: F. Rossi, V. Cardellini, F. Lo Presti, M. Nardelli,
"Geo-distributed efficient deployment of containers with Kubernetes", *Computer Communications*, Vol. 159, pp. 161-174, June 2020

ELASTIC APPLICATIONS

- Cloud computing provides resources on demand
- Dynamism of working conditions

- Calling for development of **elastic applications**
    - to change application deployment at run-time
    - to meet Quality of Service requirements

ELASTIC APPLICATIONS



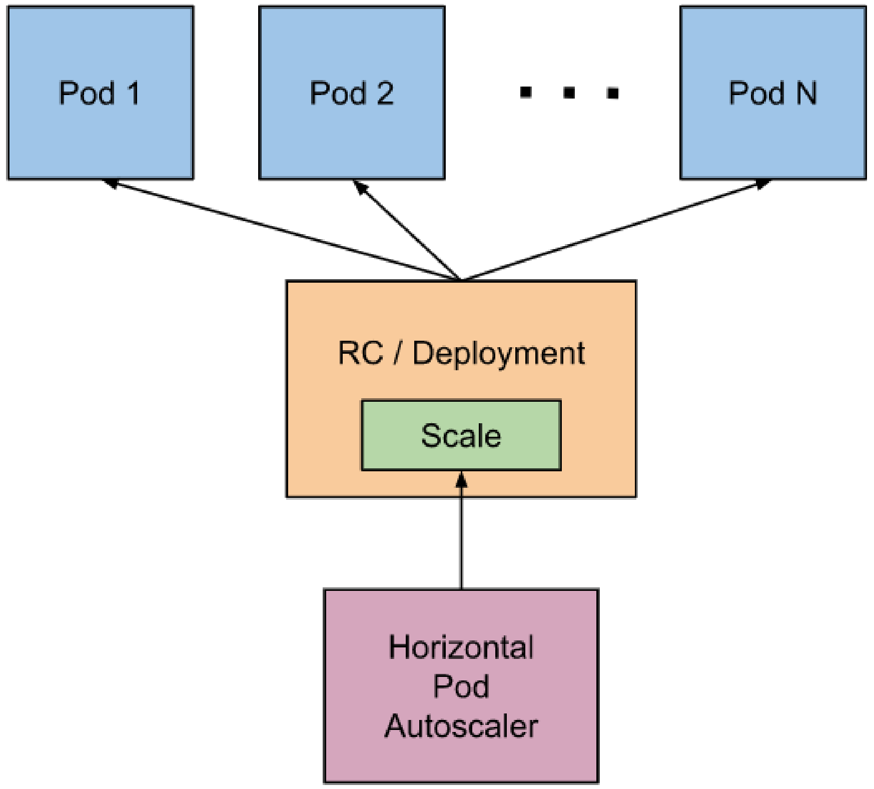
To react to workload variations

Horizontal Pod Autoscaler (HPA)

- The **Horizontal Pod Autoscaler** automatically scales the number of Pods in a deployment, replica set or stateful set based on observed CPU utilization (or, with custom metrics support, on some other application-provided metrics).
- Note that Horizontal Pod Autoscaling does not apply to objects that can't be scaled, for example, DaemonSets.

Horizontal Pod Autoscaler (HPA)

- The Horizontal Pod Autoscaler is implemented as a Kubernetes API resource and a controller.
- The resource determines the behavior of the controller.
- The controller periodically adjusts the number of replicas in a deployment to match the observed average CPU utilization to the target specified by user.



```
kubectl autoscale deployment FILENAME --cpu-percent=50 --min=1 --max=10
```

HPA: Algorithm Details

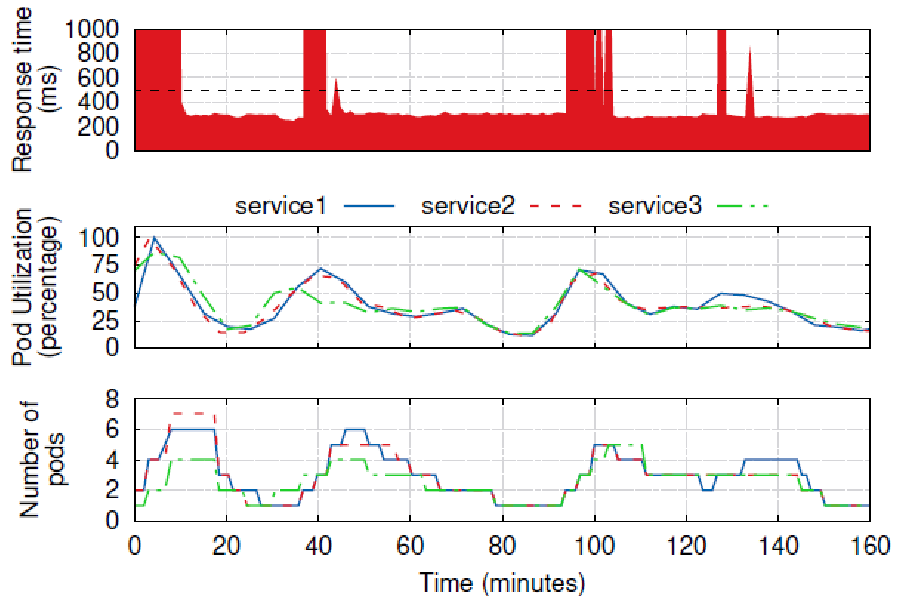
```
desiredReplicas = ceil[currentReplicas * ( currentMetricValue / desiredMetricValue )]
```

Example:

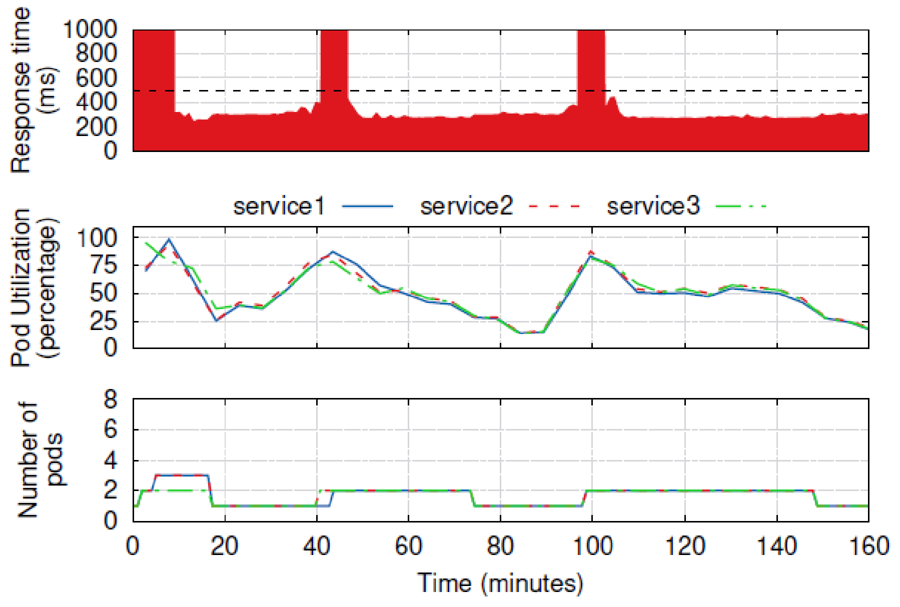
- If currentMetricValue = 200m and desiredMetricValue = 100m, the number of replicas will be double.
- If currentMetricValue = 50m and desiredMetricValue = 100m, we'll halve the number of replicas.

HPA: Algorithms Details (1)

- Tuning the (static) scaling threshold is a cumbersome task!



(a) Scaling threshold set to 50% of pod utilization

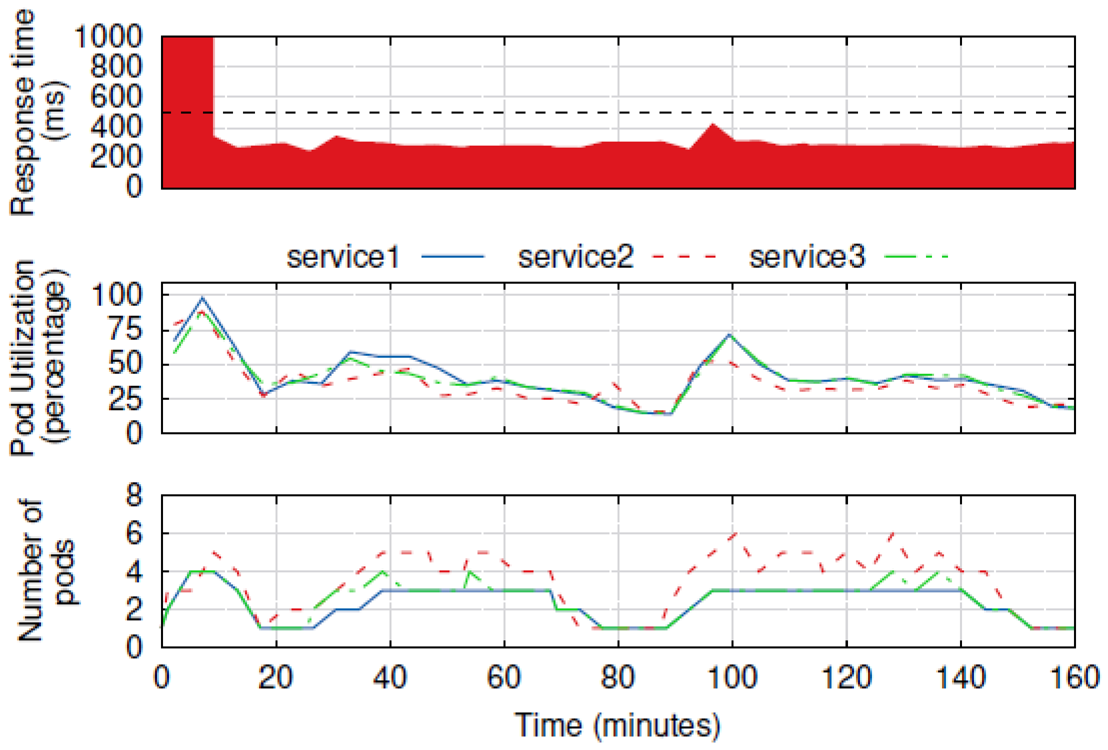


(b) Scaling threshold set to 80% of pod utilization

Source: F. Rossi, V. Cardellini, F. Lo Presti,
"Hierarchical scaling of microservices in Kubernetes", *Proceedings of the 1st IEEE International Conference on Autonomic Computing and Self- Organizing Systems (ACSOS 2020)*, Washington, DC, Washington, USA, August 17-21 2020.

HPA: Algorithms Details (2)

…but we can use more sophisticated approaches (e.g., queuing-based elasticity policy)



(a) Predictive Decision Making

Source: F. Rossi, V. Cardellini, F. Lo Presti, "Hierarchical scaling of microservices in Kubernetes", *Proceedings of the 1st IEEE International Conference on Autonomic Computing and Self- Organizing Systems (ACSOS 2020)*, Washington, DC, Washington, USA, August 17-21 2020.

Kubernetes 1.6 and HPA

- Recently, the HPA adds support for
  - multiple metrics
    - HPA controller will evaluate each metric and propose a new scale based on that metric.
    - The largest of the proposed scales will be used as the new scale.
  - making use of custom metrics
  - configurable scaling behavior
    - Behaviors are specified separately for scaling up and down
    - The stabilization window is used to restrict the flapping of replicas when the metrics used for scaling keep fluctuating

The current stable version, which only includes support for CPU autoscaling, can be found in the autoscaling/v1 API version.
The beta version, which includes support for scaling on memory and custom metrics, can be found in autoscaling/v2beta2.

References

- https://kubernetes.io/docs/home/
- https://linuxacademy.com/site-content/uploads/2019/04/Kubernetes-Cheat-Sheet_07182019.pdf
- Jawarneh, I.M.A., Bellavista, P., Bosi, F., Foschini, L., Martuscelli, G., Montanari, R., Palopoli, A.: *Container orchestration engines: A thorough functional and performance comparison.* In Proc. of IEEE ICC 2019. pp. 1–6 (2019)