WIZELINE

Cloud Native Serverless

#sre-emea

Your Questions during the session

Maybe in the session, we does not leave much time for questions.

Please post your questions to following Google doc

https://drive.google.com/file/d/15PvTVs3EBNVrUvA9TOfy37ImZ4-fQuCz/view?usp=sharing

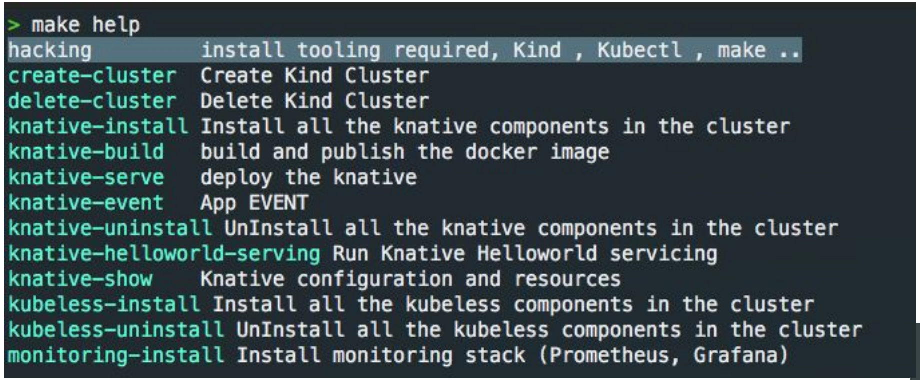
OR let´s discuss in the team Slack channel **#sre-emea**

Repo ➜ cloudnative-serverless-workshops

Agenda

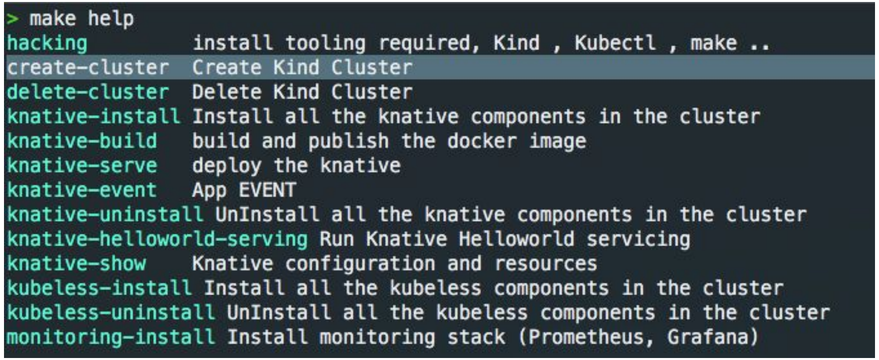
- About the Cloud native Serverless workshop series. (16th Juny Workshops)
  - Goals
  - Quick review about resources, repository
- Serverless
  - What is serverless in a nutshell?
  - Serverless In Cloud native environments (Landscape)
- Kubernetes as your foundation.
- Knative
  - Introduction
  - Workshop Serving example.

Hacking your Laptop



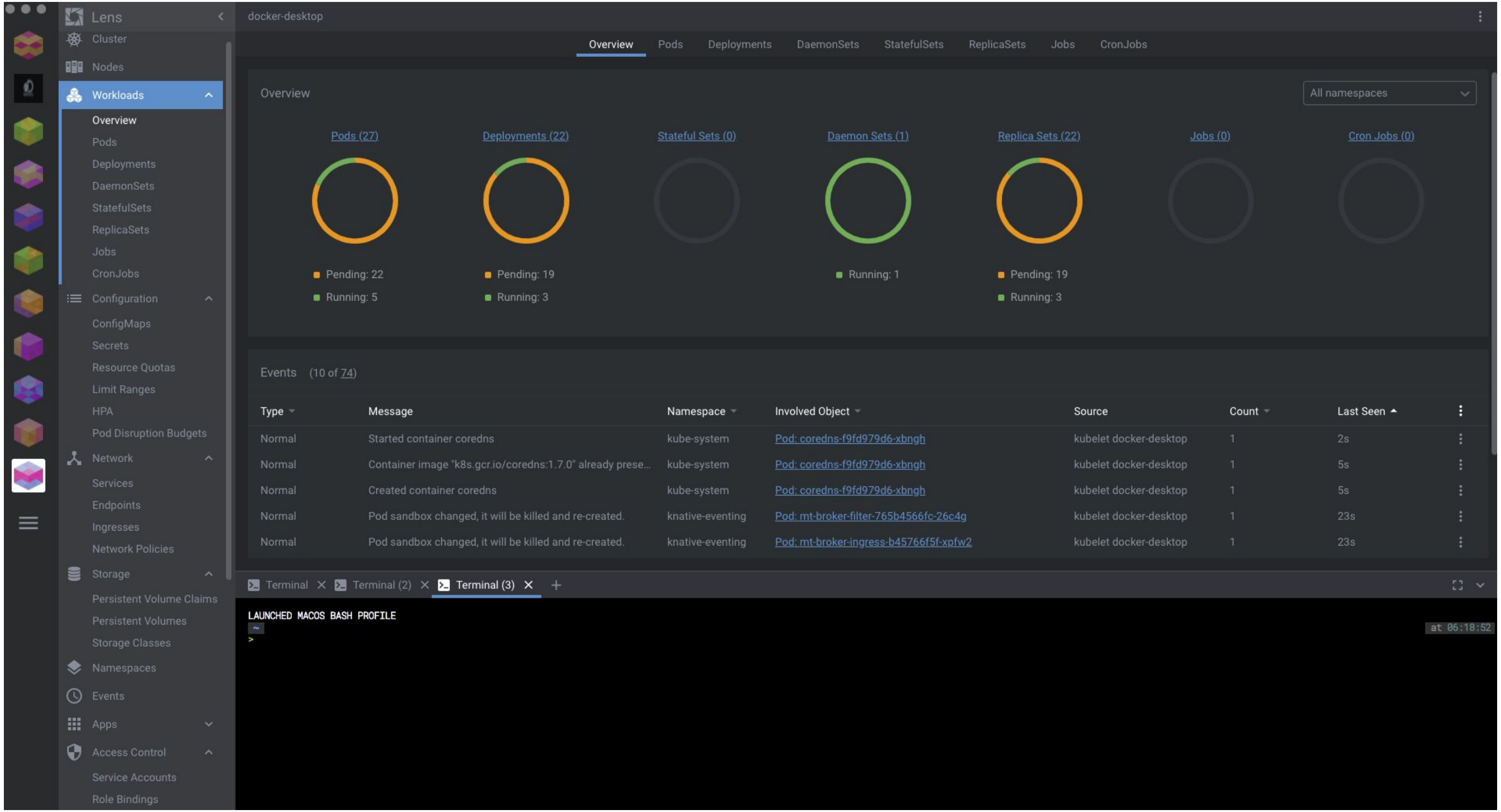
```
> make help
hacking           install tooling required, Kind , Kubectl , make ..
create-cluster    Create Kind Cluster
delete-cluster    Delete Kind Cluster
knative-install   Install all the knative components in the cluster
knative-build     build and publish the docker image
knative-serve     deploy the knative
knative-event     App EVENT
knative-uninstall UnInstall all the knative components in the cluster
knative-helloworld-serving Run Knative Helloworld servicing
knative-show      Knative configuration and resources
kubeless-install Install all the kubeless components in the cluster
kubeless-uninstall UnInstall all the kubeless components in the cluster
monitoring-install Install monitoring stack (Prometheus, Grafana)
```

Setting UP a Kubernetes Cluster



```
> make help
hacking          install tooling required, Kind , Kubectl , make ..
create-cluster   Create Kind Cluster
delete-cluster   Delete Kind Cluster
knative-install  Install all the knative components in the cluster
knative-build    build and publish the docker image
knative-serve    deploy the knative
knative-event    App EVENT
knative-uninstall UnInstall all the knative components in the cluster
knative-helloworld-serving Run Knative Helloworld servicing
knative-show     Knative configuration and resources
kubeless-install Install all the kubeless components in the cluster
kubeless-uninstall UnInstall all the kubeless components in the cluster
monitoring-install Install monitoring stack (Prometheus, Grafana)
```

Lens Kubernetes IDE



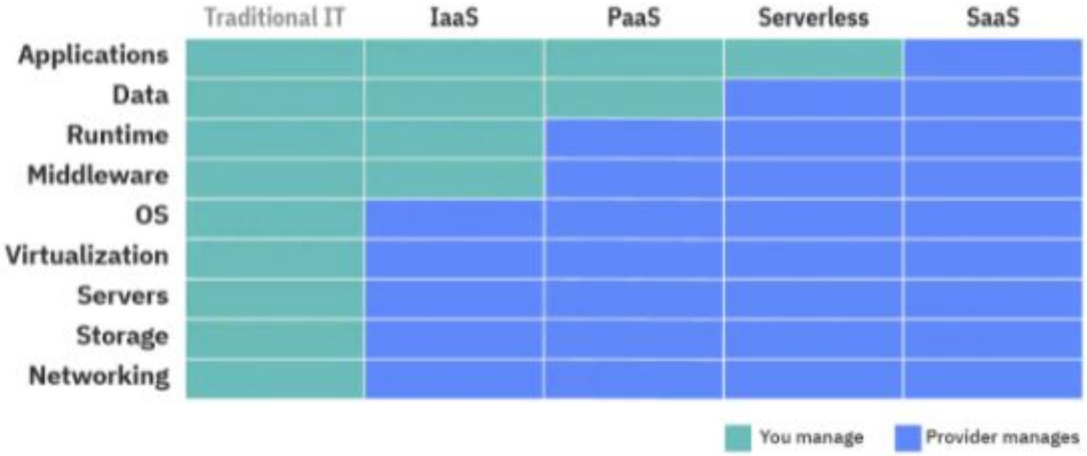
Type	Message	Namespace	Involved Object	Source	Count	Last Seen
Normal	Started container coredns	kube-system	Pod: coredns-f9fd979d6-xbngh	kubelet docker-desktop	1	2s
Normal	Container image "k8s.gcr.io/coredns:1.7.0" already prese...	kube-system	Pod: coredns-f9fd979d6-xbngh	kubelet docker-desktop	1	5s
Normal	Created container coredns	kube-system	Pod: coredns-f9fd979d6-xbngh	kubelet docker-desktop	1	5s
Normal	Pod sandbox changed, it will be killed and re-created.	knative-eventing	Pod: mt-broker-filter-765b4566fc-26c4g	kubelet docker-desktop	1	23s
Normal	Pod sandbox changed, it will be killed and re-created.	knative-eventing	Pod: mt-broker-ingress-b45766f5f-xpfw2	kubelet docker-desktop	1	23s

```
LAUNCHED MACOS BASH PROFILE
 ~                                                                  at 06:18:52
>
```

WIZELINE

Serverless

What is Serverless ?



	Traditional IT	IaaS	PaaS	Serverless	SaaS
Applications	🟩	🟩	🟩	🟩	🟦
Data	🟩	🟩	🟩	🟦	🟦
Runtime	🟩	🟩	🟦	🟦	🟦
Middleware	🟩	🟩	🟦	🟦	🟦
OS	🟩	🟦	🟦	🟦	🟦
Virtualization	🟩	🟦	🟦	🟦	🟦
Servers	🟩	🟦	🟦	🟦	🟦
Storage	🟩	🟦	🟦	🟦	🟦
Networking	🟩	🟦	🟦	🟦	🟦

🟩 You manage 🟦 Provider manages

What is Serverless ?

A unit of work that consumes resources only when it´s used (A kind of workload)

- **Function** is the unit of work
  - Stateless
  - Serves one goal. Single purposed
  - Arguments (inputs) and result (output)

Code centric paradigm. Hiding infrastructure

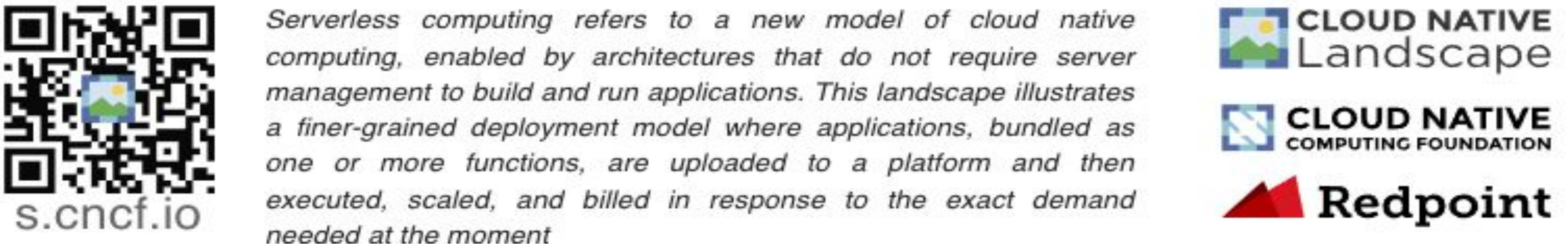
- Focus on **coding resolving business problems** and assuming that everything is working on some “computer resource”.
- User experience for **(Developers)**

What is Serverless ?

Orchestration of independent pieces of work (functions as a service)

- Carrying state of the entire flow (program)
- Error handling
- Transaction management
- Don´t pay for idle time

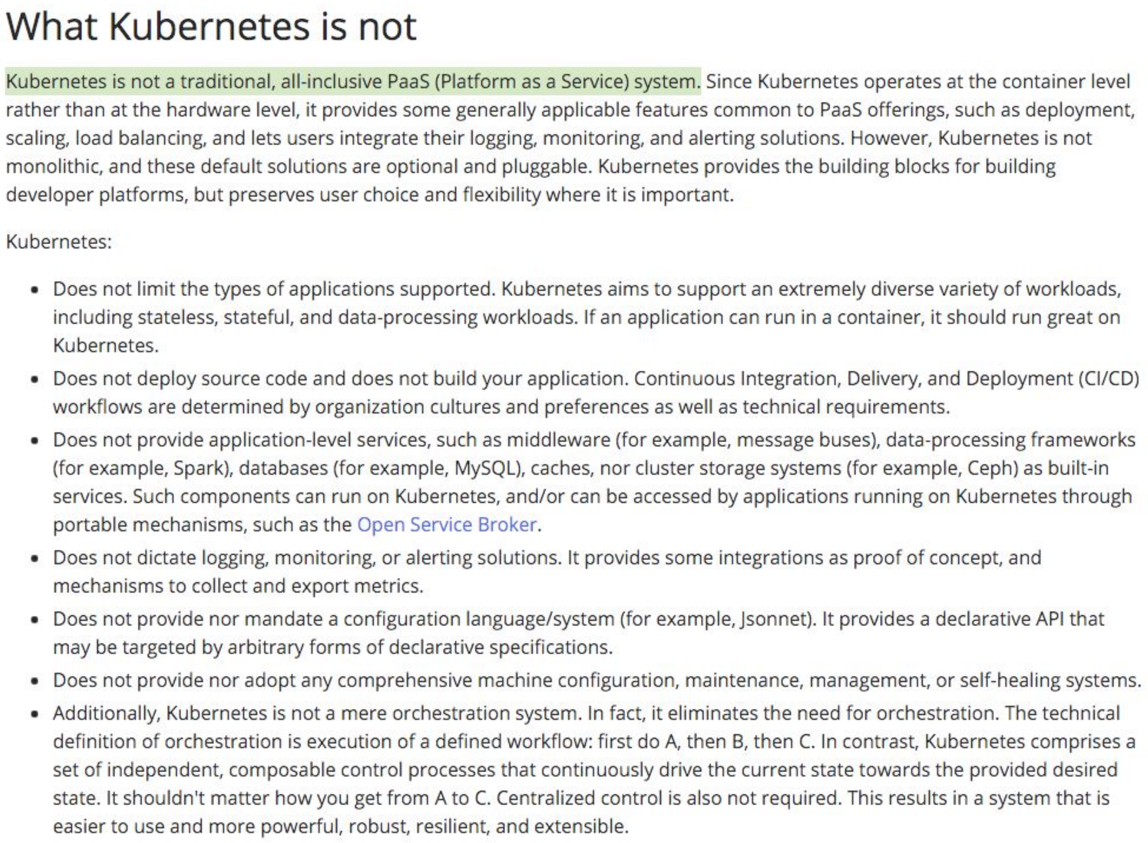
https://landscape.cncf.io/serverless



Serverless computing refers to a new model of cloud native computing, enabled by architectures that do not require server management to build and run applications. This landscape illustrates a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and billed in response to the exact demand needed at the moment

Kubernetes as Platform

What is not Kubernetes ?



What Kubernetes is not

Kubernetes is not a traditional, all-inclusive PaaS (Platform as a Service) system. Since Kubernetes operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, and lets users integrate their logging, monitoring, and alerting solutions. However, Kubernetes is not monolithic, and these default solutions are optional and pluggable. Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

Kubernetes:

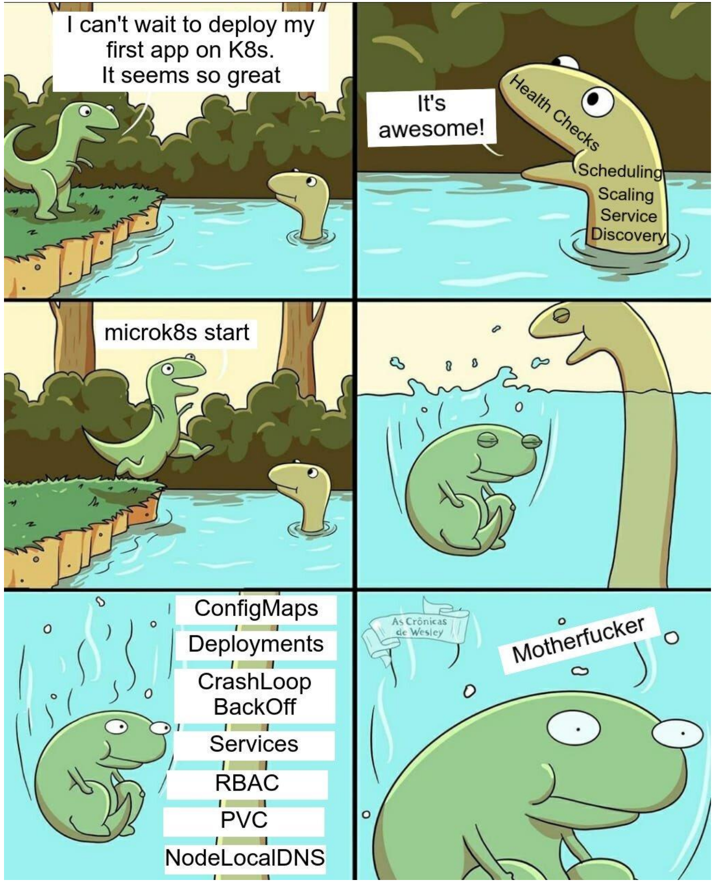
- Does not limit the types of applications supported. Kubernetes aims to support an extremely diverse variety of workloads, including stateless, stateful, and data-processing workloads. If an application can run in a container, it should run great on Kubernetes.
- Does not deploy source code and does not build your application. Continuous Integration, Delivery, and Deployment (CI/CD) workflows are determined by organization cultures and preferences as well as technical requirements.
- Does not provide application-level services, such as middleware (for example, message buses), data-processing frameworks (for example, Spark), databases (for example, MySQL), caches, nor cluster storage systems (for example, Ceph) as built-in services. Such components can run on Kubernetes, and/or can be accessed by applications running on Kubernetes through portable mechanisms, such as the Open Service Broker.
- Does not dictate logging, monitoring, or alerting solutions. It provides some integrations as proof of concept, and mechanisms to collect and export metrics.
- Does not provide nor mandate a configuration language/system (for example, Jsonnet). It provides a declarative API that may be targeted by arbitrary forms of declarative specifications.
- Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems.
- Additionally, Kubernetes is not a mere orchestration system. In fact, it eliminates the need for orchestration. The technical definition of orchestration is execution of a defined workflow: first do A, then B, then C. In contrast, Kubernetes comprises a set of independent, composable control processes that continuously drive the current state towards the provided desired state. It shouldn't matter how you get from A to C. Centralized control is also not required. This results in a system that is easier to use and more powerful, robust, resilient, and extensible.

Is Kubernetes a Platform ?

Kubernetes is not your platform, is your Foundation

Ref → Manuel Paris, Kubernetes Is Not Your Platform, It's Just the Foundation

Is Kubernetes a Platform ?



What is Container as a Function ?

We can assume that is the same than serverless:

- Stateless
- Serves one goal. Single purposed
- Arguments (inputs) and result (output)

AND

Static self-running piece of work wrapped into a container with everything it needs for its works)

- Code + **Platform**

Knative

Knative

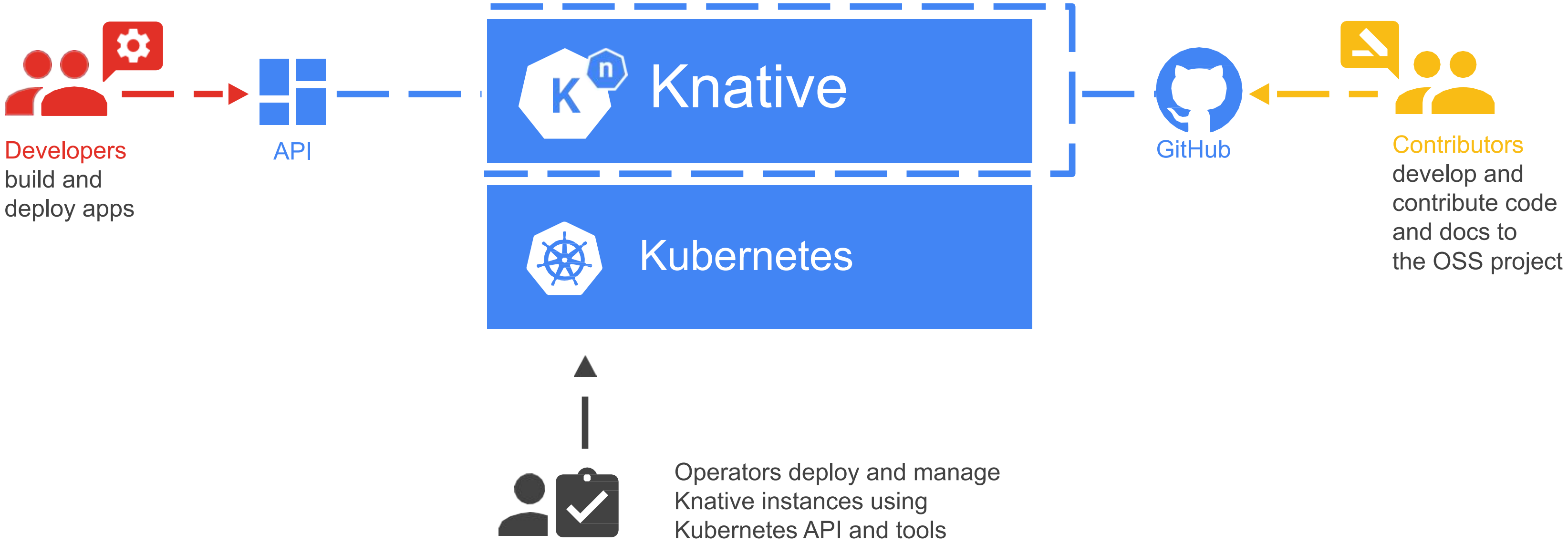
Knative, **created by Google** with contributions from more than 50 different companies, delivers an essential set of components to build and run serverless applications on Kubernetes.

Knative components focus on solving mundane but difficult tasks such as deploying a container, routing and managing traffic with blue/green deployment, scaling automatically and sizing workloads based on demand, and binding running services to eventing ecosystems. **The Google Cloud Run service is built from Knative.**

What is Knative?

- A **Platform** installed on top of Kubernetes that brings the capabilities of serverless
  - Kubernetes workloads ➙ Serverless Workloads
  - Building Blocks Components:
    - Build : Code ➙ Container ➙ Kubernetes
    - Serving: Route (Scaling Routing) + Configuration (Snapshots)
    - Eventing: Triggers + Pipelines

Knative

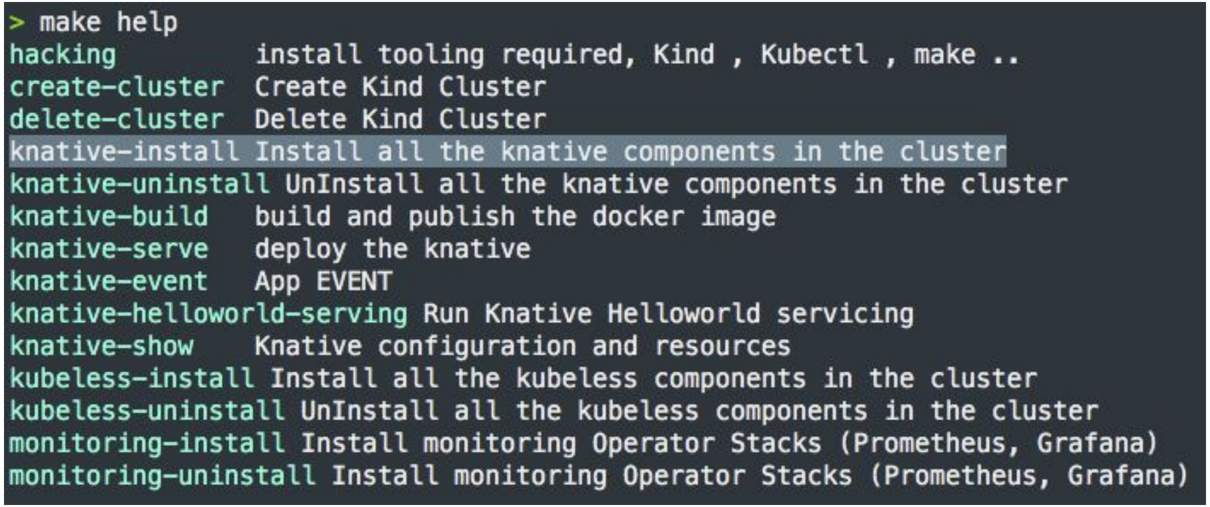


Developers build and deploy apps

Contributors develop and contribute code and docs to the OSS project

Operators deploy and manage Knative instances using Kubernetes API and tools

Install Knative in Kubernetes



```
> make help
hacking         install tooling required, Kind , Kubectl , make ..
create-cluster  Create Kind Cluster
delete-cluster  Delete Kind Cluster
knative-install Install all the knative components in the cluster
knative-uninstall UnInstall all the knative components in the cluster
knative-build   build and publish the docker image
knative-serve   deploy the knative
knative-event   App EVENT
knative-helloworld-serving Run Knative Helloworld servicing
knative-show    Knative configuration and resources
kubeless-install Install all the kubeless components in the cluster
kubeless-uninstall UnInstall all the kubeless components in the cluster
monitoring-install Install monitoring Operator Stacks (Prometheus, Grafana)
monitoring-uninstall Install monitoring Operator Stacks (Prometheus, Grafana)
```

Knative Serving

Service: The **service.serving.knative.dev** resource automatically manages the whole lifecycle of your workload. It controls the creation of other objects to ensure that your app has a route, a configuration, and a new revision for each update of the service. Service can be defined to always route traffic to the latest revision or to a pinned revision.

Route: **The route.serving.knative.dev** resource maps a network endpoint to one or more revisions. You can manage the traffic in several ways, including fractional traffic and named routes.

Knative Serving

Configuration: The **configuration.serving.knative.dev** resource maintains the desired state for your deployment. It provides a clean separation between code and configuration and follows the Twelve-Factor App methodology. Modifying a configuration creates a new revision:

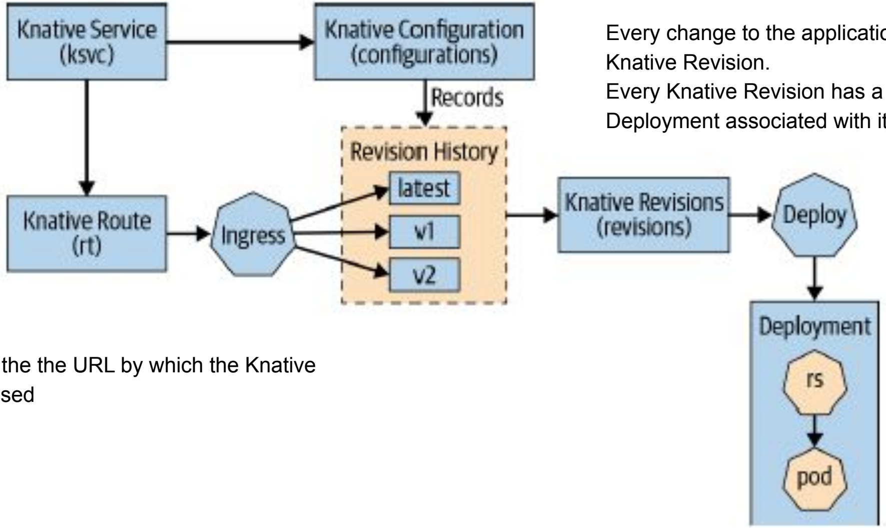
- Revision: The `revision.serving.knative.dev` resource is a point-in-time snapshot of the code and configuration for each modification made to the workload. Revisions are immutable objects and can be retained for as long as useful. Knative Serving Revisions can be automatically scaled up and down according to incoming traffic. See Configuring the Autoscaler for more information.

Understanding Knative Serving

Knative Service is the main Controller

Based on the desired state, the Knative Configuration controller creates a new Kubernetes Deployment for your application.

Every change to the application configuration creates a new Knative Revision.
Every Knative Revision has a corresponding Kubernetes Deployment associated with it



The Knative Route is the the URL by which the Knative Service can be accessed
or invoked.

https://knative.dev/docs/serving/

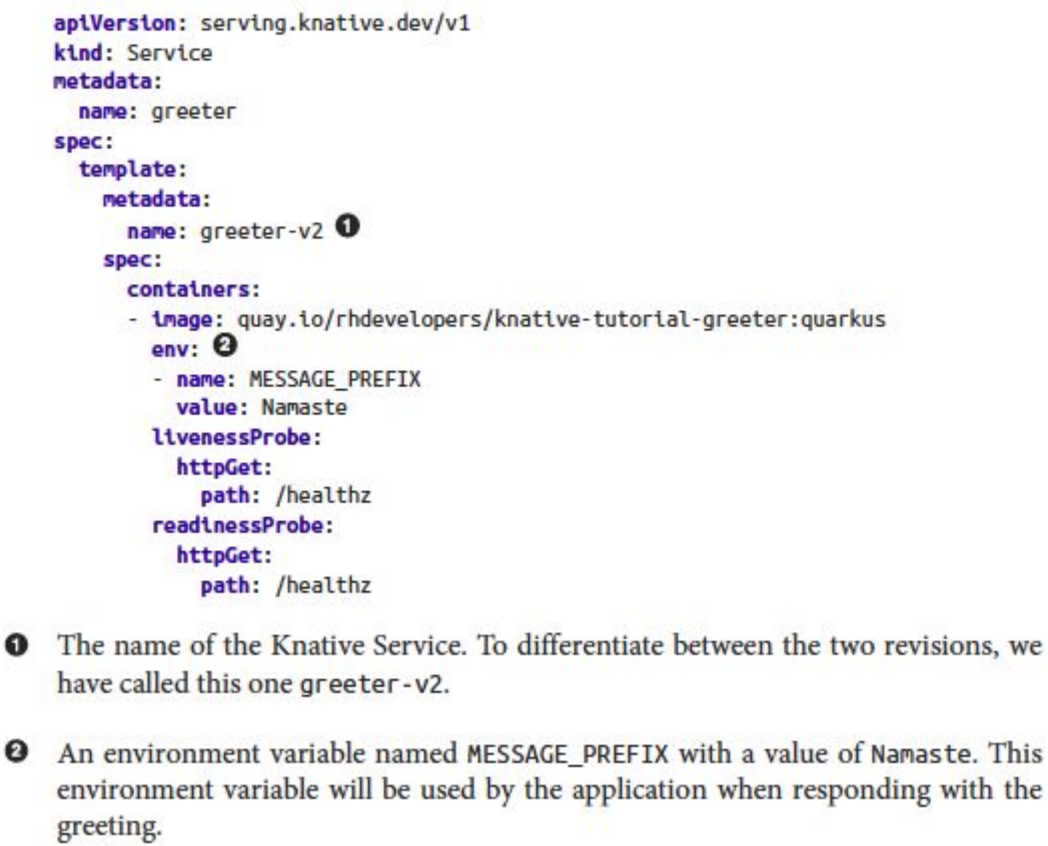
Knative Serving

```
kubectl get route <route-name> --output yaml
```

```
kubectl get configuration <configuration-name> --output
jsonpath="{.status.latestCreatedRevisionName}"
```

```
kubectl get revision <revision-name> --output yaml
```

Updating a Knative Service Configuration



```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: greeter
spec:
  template:
    metadata:
      name: greeter-v2 ❶
    spec:
      containers:
      - image: quay.io/rhdevelopers/knative-tutorial-greeter:quarkus
        env: ❷
        - name: MESSAGE_PREFIX
          value: Namaste
        livenessProbe:
          httpGet:
            path: /healthz
        readinessProbe:
          httpGet:
            path: /healthz
```

❶ The name of the Knative Service. To differentiate between the two revisions, we have called this one `greeter-v2`.

❷ An environment variable named `MESSAGE_PREFIX` with a value of `Namaste`. This environment variable will be used by the application when responding with the greeting.

Knative Serving. Autoscaling

Scale-to-zero (Activator)

- After a time, your Kn service revision is considered to be inactive.
  - Knative will terminate all the pods
  - Routes for the inactive Revision well be mapped by Activator service
  - Arguments (inputs) and result (output)

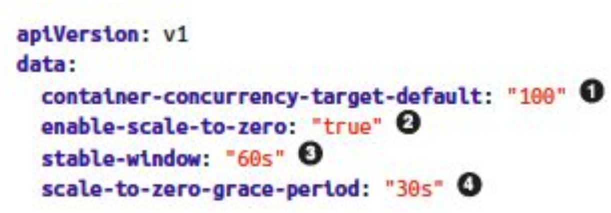
Autoscaling (Autoscaler)

- The Knative service scale out, its based on inbound HTTP traffic.Focus on **coding resolving business problems** and assuming that everything is working on some "computer resource".
  - Knative Horizontal Pod Autoscaler (KPA)
  - Horizontal Pod Autoscaler (HPA). This one is the default Kubernetes autoscaler method. (concurrency, request/sc, cpu)

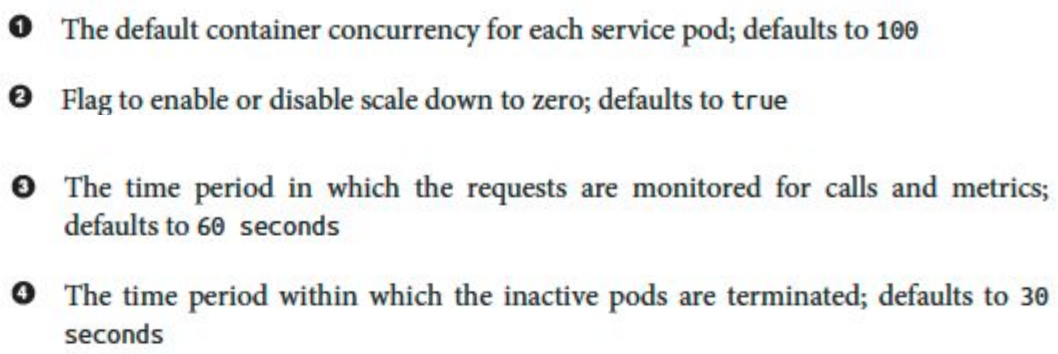
https://knative.dev/docs/serving/autoscaling/

Knative Serving. Autoscaling Configuration

```
kubectl -n knative-serving get cm
config-autoscaler -o yaml
```



```
apiVersion: v1
data:
  container-concurrency-target-default: "100" ❶
  enable-scale-to-zero: "true" ❷
  stable-window: "60s" ❸
  scale-to-zero-grace-period: "30s" ❹
```



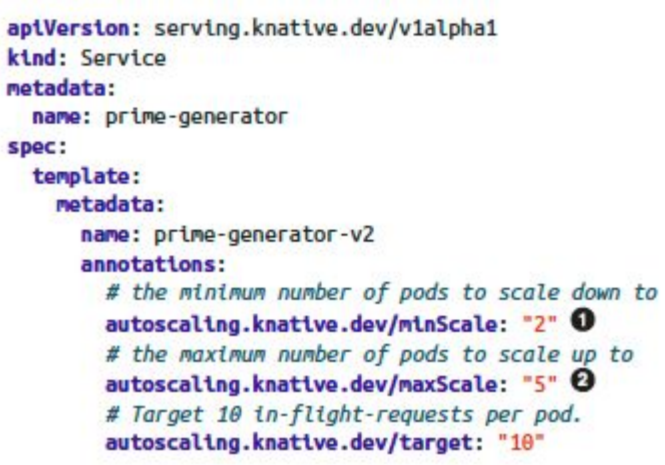
❶ The default container concurrency for each service pod; defaults to `100`

❷ Flag to enable or disable scale down to zero; defaults to `true`

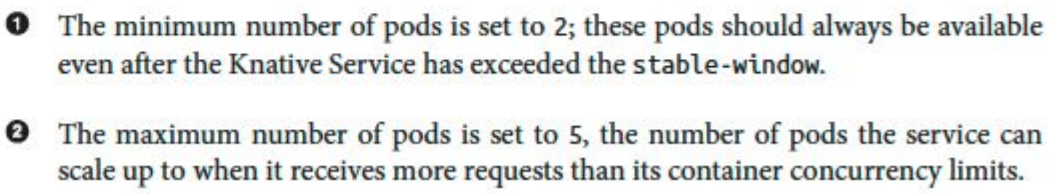
❸ The time period in which the requests are monitored for calls and metrics; defaults to `60 seconds`

❹ The time period within which the inactive pods are terminated; defaults to `30 seconds`

Knative Serving. Autoscaling minScale - maxScale



```yaml
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: prime-generator
spec:
  template:
    metadata:
      name: prime-generator-v2
      annotations:
        # the minimum number of pods to scale down to
        autoscaling.knative.dev/minScale: "2" ❶
        # the maximum number of pods to scale up to
        autoscaling.knative.dev/maxScale: "5" ❷
        # Target 10 in-flight-requests per pod.
        autoscaling.knative.dev/target: "10"
```



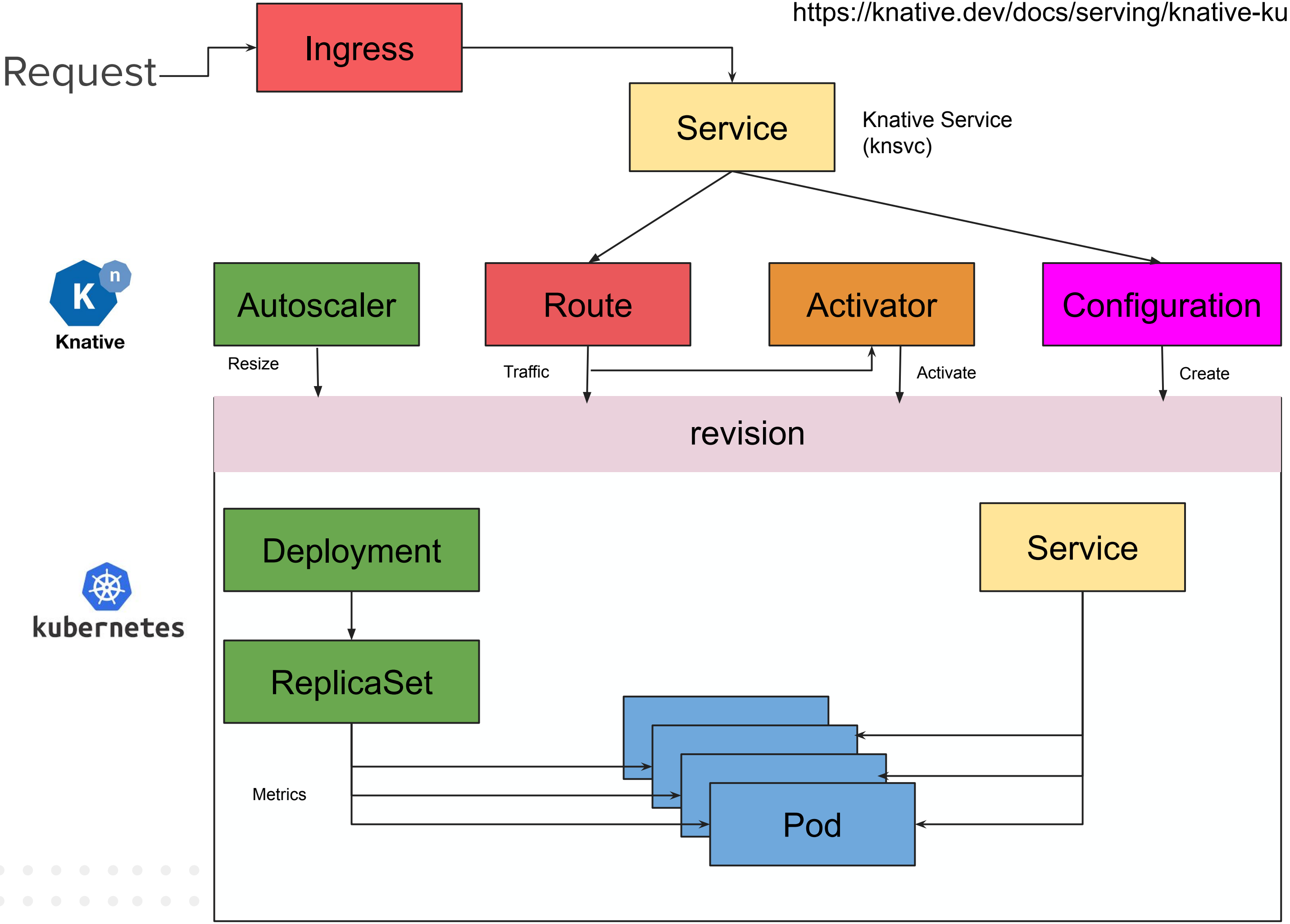
❶ The minimum number of pods is set to 2; these pods should always be available even after the Knative Service has exceeded the `stable-window`.

❷ The maximum number of pods is set to 5, the number of pods the service can scale up to when it receives more requests than its container concurrency limits.

Knative Autoscaling Sample

https://knative.dev/docs/serving/autoscaling/autoscale-go/

https://knative.dev/docs/serving/knative-kubernetes-services/


Request
Ingress
Service
Knative Service
(knsvc)
Knative
Autoscaler
Route
Activator
Configuration
Resize
Traffic
Activate
Create
revision
kubernetes
Deployment
ReplicaSet
Service
Metrics
Pod