A small introduction to Kubernetes

Dr. Guillaume Rosinosky

guillaume.rosinosky@uclouvain.be

UCL
Université
catholique
de Louvain

Introduction

- Container orchestration systems
  - Small scale: Docker Swarm / Compose (mono node)
- Medium/Large scale: Kubernetes (K8S)
  - Origin: Google, helmsman or pilot in Greek
  - Google's Borg, Omega (closed-source)
  - And then Kubernetes open-sourced in 2014
  - Wide usage and support
- Supported by Cloud Native Computing Foundation (CNCF) – Linux Foundation project including Google, Twitter, Intel, Cisco, IBM ...
- 2020 CNCF Survey : 78 % of respondants use K8S in production
- Provided by all cloud « big-players » EKS (AWS), AKS (Azure), GKE (Google), IBM, Alibaba…

Outline

- Introduction
- Architecture
- Basic objects (deployments, services, storage, configuration)
- Demo Scapp
- Advanced objects (jobs, elasticity)
- Demo elasticity on Scapp
- Installation and package management
- Helm Demo on WordPress
- Extension and tooling
- Demo Prometheus
- Conclusion

Main features

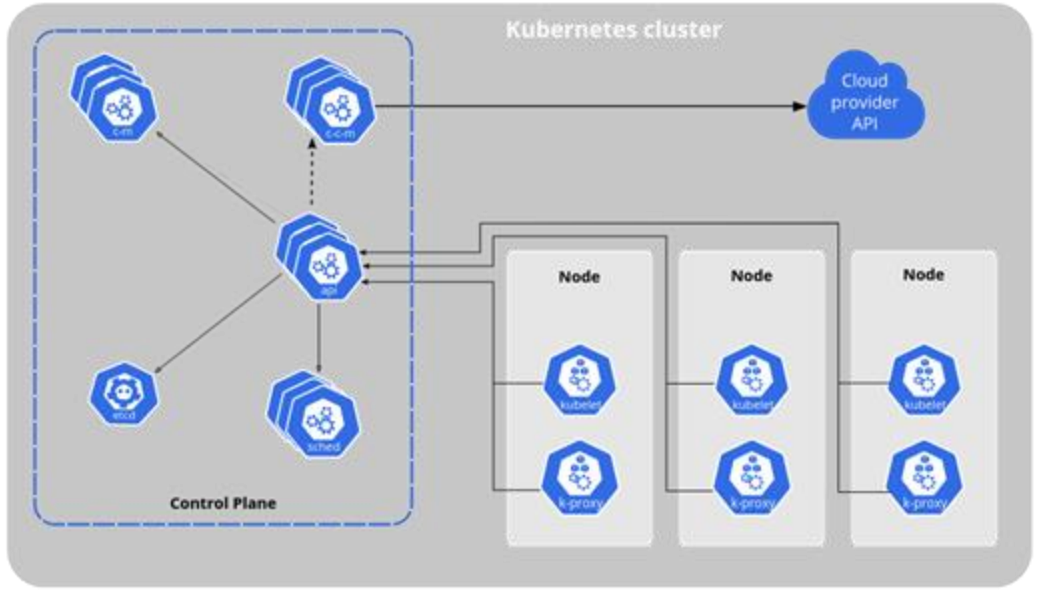
- Service discovery and load balancing
  - Traffic distribution + DNS abstraction in front of containers
- Automated rollout and rollbacks
  - Launch set of operations for declared state at a controlled rate
- Automated bin-packing
  - Scheduling based on requirements/limits for containers vs node capacity
- Self-healing
  - Set number of replicas, cluster automatically creates/removes containers based on health-check
- Secrets and configuration management
  - Management of configuration files and passwords/tokens/keys
- Storage orchestration
  - Integration with cloud storage/local storage
- Cloud-ready
  - Connectors for most public Cloud providers (network, persistence)
- Extensibility
  - Well defined APIs, huge ecosystem

Basic support in Docker Swarm

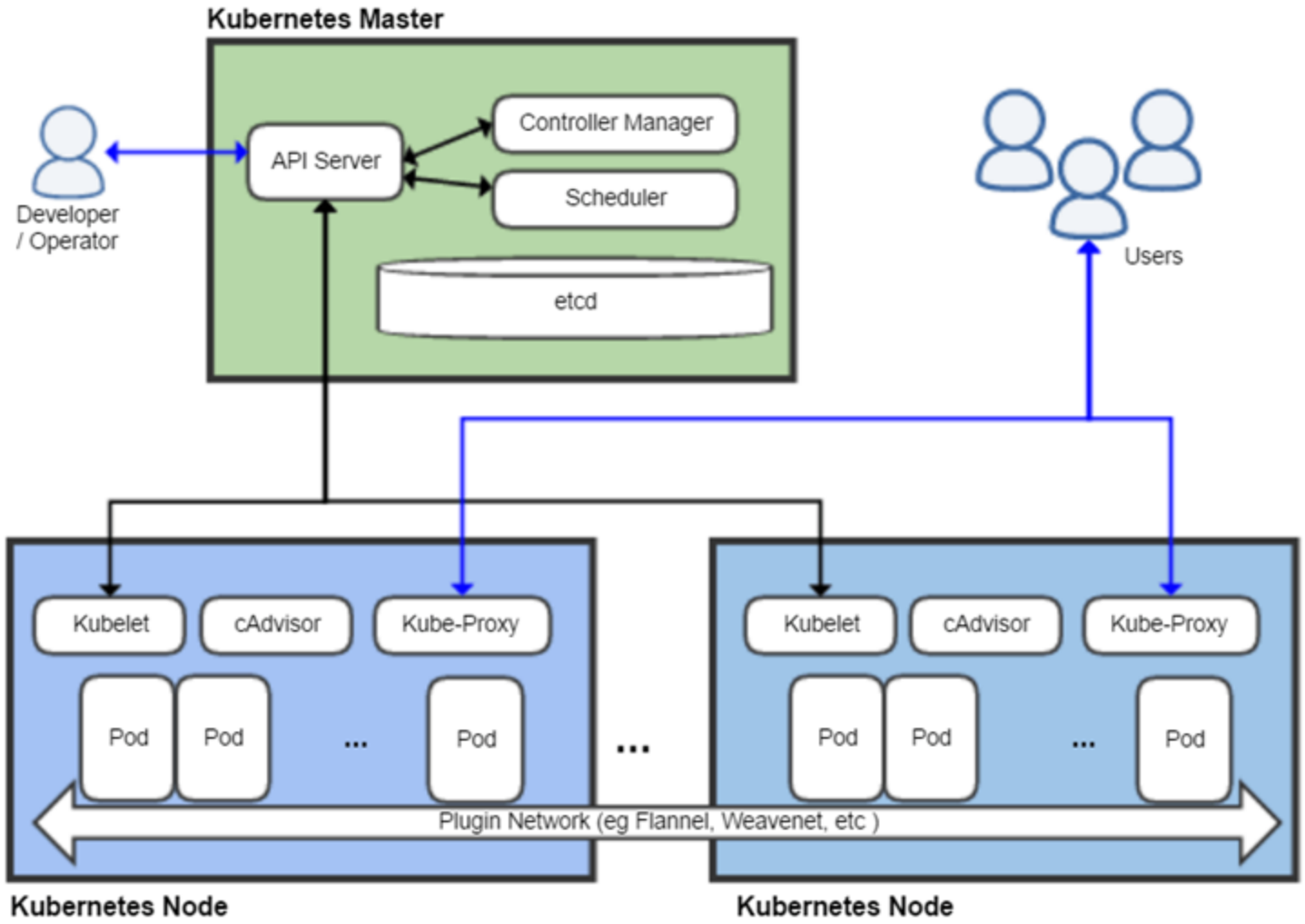
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes server modules

- Control Plane (control/"master" nodes)
  - kube-apiserver (HTTP/HTTPS API)
  - etcd (KV database)
  - kube-scheduler (containers scheduling on nodes)
  - kube-controller-manager (management, nodes, containers replication, endpoints, service accounts)
  - cloud-controller-manager
- Nodes components (worker nodes)
  - kubelet (container management and telemetry)
  - kube-proxy (load balancing and routing)
  - container-runtime (Docker, containerD, CRIO etc.)



https://kubernetes.io/docs/concepts/overview/components/


Kubernetes Master
Developer
/ Operator
API Server
Controller Manager
Scheduler
etcd
Users
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
...
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Plugin Network (eg Flannel, Weavenet, etc)
Kubernetes Node
Kubernetes Node

Kubernetes objects

- Declarative K8S object configuration in YAML description file
  - As in Docker Compose/Swarm, you declare the state you expect, and the cluster does the operations
- State of cluster manipulated with persisted K8S objects in etcd sharing the following attributes :
  - apiVersion: version of k8s API
  - kind: type of object
  - metadata: name, labels
  - spec: object details
- Usable tools/API for imperative commands

https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

Basic element : Pod

- Group of containers sharing Linux namespaces, network, and volumes
- Containers of a pod are deployed on the same physical node
- As in docker-compose: ports, environment variables, command to launch, storage/files mount
- Init containers

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
```

https://kubernetes.io/docs/concepts/workloads/pods/

Deployments

- Pods are a one-time deployment … what about High Availability (HA) and scalability?
- Deployment
  - Set of pods for long-running stateless servers
  - Adds High Availability
  - Possibility to scale out/in number of replicas
  - Random names

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Services

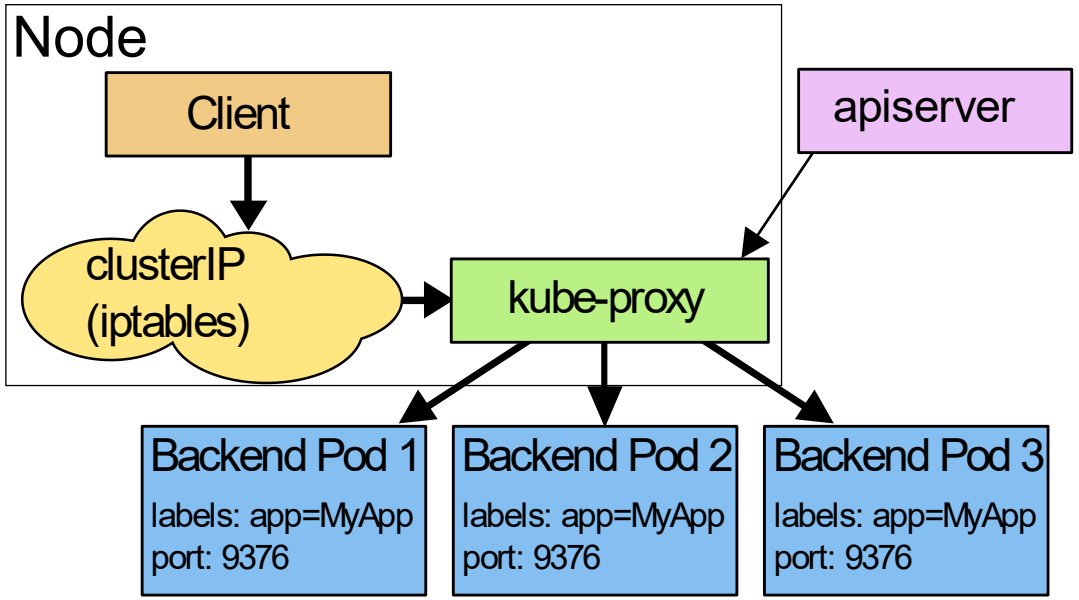
- How do we make communications with pods?
- Services
  - TCP/UDP proxy (kube-proxy) + iptables / IP Virtual Server
  - Main types
    - ClusterIP (incluster) - default
    - Headless (no routing, direct access to pods)
    - NodePort (linked to nodes port 30000-32767)
    - LoadBalancer (external Load Balancer: cloud or metalLB)
  - Load balancing: round-robin, random, least connection, hashing, …

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

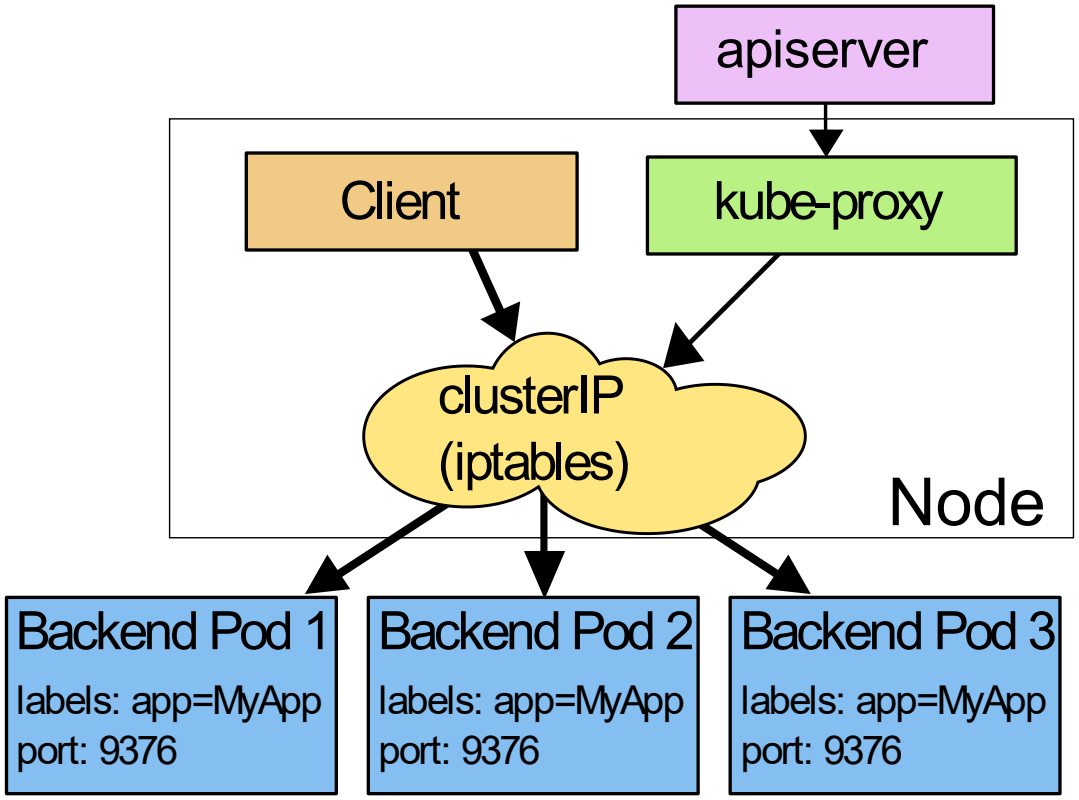
```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx
```

https://kubernetes.io/docs/concepts/services-networking/service/

Services with iptables



User space proxy mode



Iptables proxy mode

Persistence

- How to ensure access to long term storage ?
- Storage class
  - Provisioner and storage type
  - Cloud, user-defined
- PersistentVolumeClaim (PVC)
  - User's demand for storage
  - Parameters: storage size, Read/Write mode, etc.
  - Set in pods / deployments
- PersistentVolume (PV)
  - Actual storage abstraction
  - Static on node (Persistent Volume with path and nodeSelector for instance)
  - Dynamic (Network FileSystem (NFS), Cloud storage…)

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: claim1
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: fast
  resources:
    requests:
      storage: 30Gi
```

https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/

Statefulset

- What about non stateless applications ?
- Statefulset
  - Deployment for stateful apps
    - PVC template
    - Pod identity : 0, 1, 2…
    - Sequential deployment
  - Used for databases, batch/stream processing, etc.

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

Ingress

- How to manage external access ?
- Ingress
  - Reverse proxy
  - Simple HTTP/HTTPS traffic routing versus services
  - TLS/SSL encryption
  - Possible to use supported reverse proxies:
    - nginx
    - traefik
    - …

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-wildcard-host
spec:
  rules:
  - host: "foo.bar.com"
    http:
      paths:
      - pathType: Prefix
        path: "/bar"
        backend:
          service:
            name: service1
            port:
              number: 80
  - host: "*.foo.com"
    http:
      paths:
      - pathType: Prefix
        path: "/foo"
        backend:
          service:
            name: service2
            port:
              number: 80
```

https://kubernetes.io/docs/concepts/services-networking/ingress/

Configmaps and secrets

- How to manage configuration files?
  - Configmaps
    - Environment variables or embedded configuration files mounted in Pods
  - Secrets
    - Config maps logic + encrypted/restricted access to values

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:52:05Z
  name: game-config
  namespace: default
  resourceVersion: "516"
  uid: b4952dc3-d670-11e5-8cd0-68f728db1985
data:
  game.properties: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
```

https://kubernetes.io/docs/concepts/configuration/configmap/
https://kubernetes.io/docs/concepts/configuration/secret/

Kubernetes clients

- How to control our cluster ?
- Kubectl
  - Standard command line tool for control of the cluster
  - Use ~/.kube/config file for access to clusters
  - Very powerful
- Web dashboard
- Libraries using HTTPS API access on all kind of languages (Python, Java, node.js, etc.)
- VsCode, IntelliJ extensions
- Gitops (Flux, …)

- kubectl <command> <K8S object> <parameters>
  - Commands
    - get, describe, delete, logs
  - K8S object
    - pod, deployment, statefulset, pv …
  - Ex: kubectl logs pod/couchdb-0
- kubectl apply –f <file(s)/directory>
  - Creates/update depending on YAML files
- kubectl port-forward (access to a port inside cluster)
- kubectl scale <deployment/statefulset> --replicas=N (scaling)
- kubectl exec –it pod/mypod /bin/bash *(attach shell)*
- kubectl top pods (get cpu/memory usage)
- kubectl patch <K8S object> <parameters>
- …

https://kubernetes.io/docs/reference/kubectl/overview/

Demo: Nginx then Scapp in Kubernetes

- Modules
  - couchDB (database)
  - auth-service (micro-service)
  - scapp-fe (frontend)
- Deployment of the application with kubectl
- Check results on browser and with VsCode K8S extension

- K8S Objects
  - PersistentVolume
    - couchDB
  - Statefulset
    - couchDB
  - Deployments
    - auth-service
    - scapp-fe
  - ConfigMap
  - Services
    - couchDB
    - auth-service
    - scapp-fe
- Already present
  - Kind Kubernetes installation

Other pod-based objects

- Jobs: tasks expected to terminate
  - Launch short/long time process
  - Useful attributes
    - parallelism (= replicas in Deployment/Statefulset)
    - activeDeadlineSeconds (duration)
    - backoffLimit (number of pod crashes before setting job as failed)
- Cronjobs
  - Add scheduling capability to jobs
- Daemonset
  - Launch a pod on each node (or a set of nodes)
  - Useful for cross-cluster modules (logging, service mesh, …)

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignu
      restartPolicy: Never
  backoffLimit: 4
```

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

https://kubernetes.io/docs/concepts/workloads/controllers/job/

https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

Nodes and selectors

- Nodes
  - Name
  - Status
    - Ready
    - DiskPressure, MemoryPressure…
  - Labelable
  - Capacity metrics
    - CPU / RAM / disk / network
    - Check versus usage of pods
- Add/remove nodes dynamically with management tools
  - Easy with managed solutions, more complicated on premises

- Pod
  - Possibility to « assign » pods to nodes

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/
https://kubernetes.io/docs/concepts/architecture/nodes/

Pod scheduling

- How to be sure one Pod does not eat all memory/CPU ?
- Requests and limits for CPU and memory
  - Requests
    - Used for scheduling, but not enforced
    - Scheduler uses node capabilities
  - Limits
    - Enforced via throttling (CPU) or killing (memory)
  - You should define it ! Can break a node if not.
- Affinity / anti-affinity between pods or between nodes and pods
- Scheduler
  - kube-scheduler
    - For each new pod: filtering, scoring, and then placement by kube-controller
  - Better schedulers (Machine Learning jobs, batch, stream): Volcano, Poseidon/Firmament

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

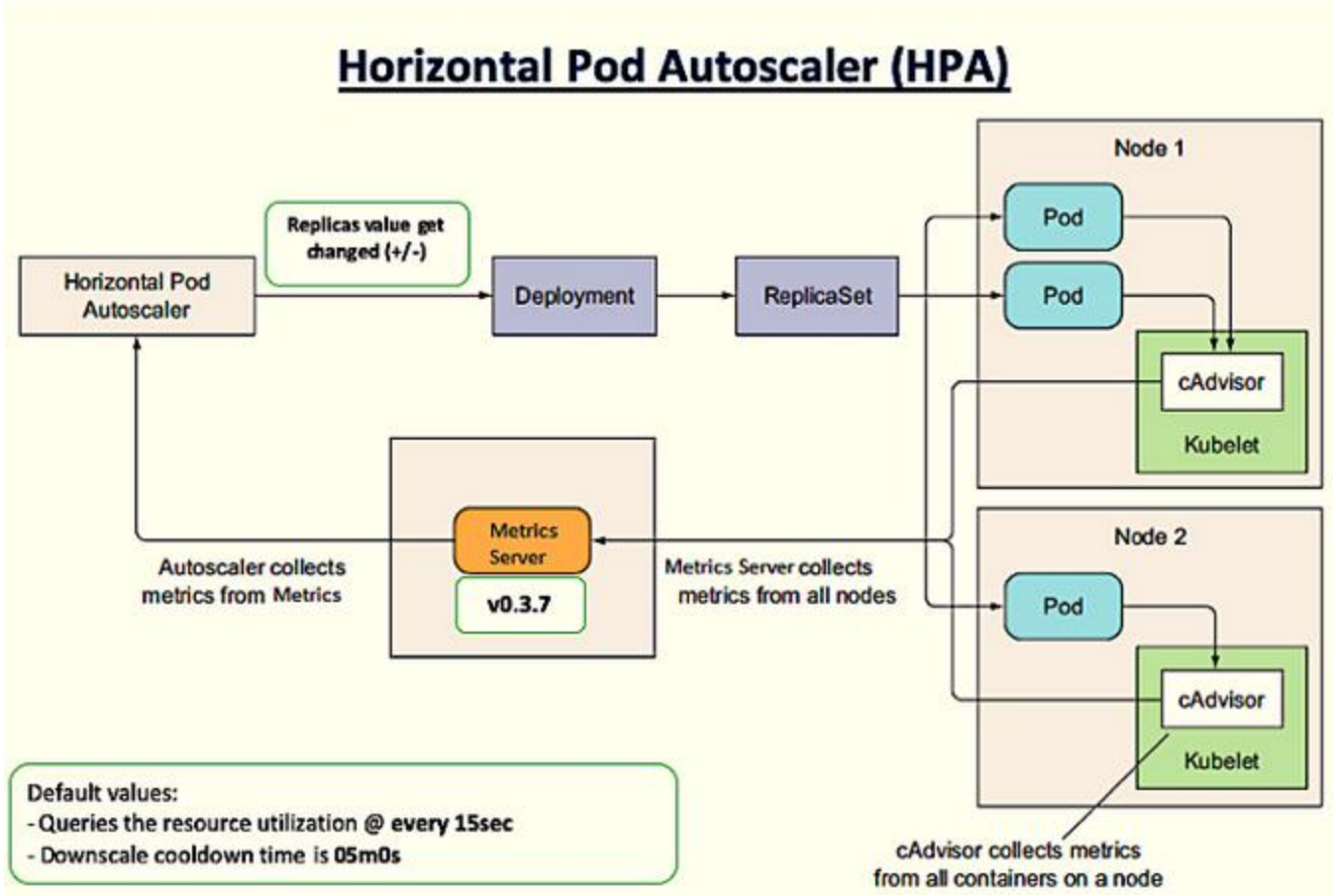
Elasticity

- Scale in/out and scale up/down ?
- Horizontal Pod Autoscaler (HPA)
  - Setting number of replicas of deployment
  - Default rule: `desiredReplicas = ceil[currentReplicas * ( currentMetricValue / desiredMetricValue )]`
- Vertical Pod Autoscaler
  - Setting pod resources requests
  - Based on past and current usage (regression)
  - Can kill pods for them to be rescheduled
- Cluster Autoscaler (cloud) :
  - Add or removes nodes depending on cluster needs

- Metrics-server
  - Basic monitoring stack / source of data for autoscalers
  - Uses cAdvisor via Kubelet module to get metrics on pods/nodes
    - CPU, memory
  - Possible to use more advanced monitoring stacks such as Prometheus

https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler
https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler

Monitoring with Horizontal Pod Autoscaler



https://shahbhargav.medium.com/horizontal-pod-autoscaler-hpa-in-kubernetes-f72917679528

Demo scalability

- The same as in tutorial with the Scapp app … but in K8S with Horizontal Pod Autoscaler.
- Load injection for 1 minute.

- Used K8S Objects
  - Job: Artillery based load tester
  - ConfigMap for load tester configuration
  - HorizontalPodAutoscaler (30% average CPU)
- Already present
  - Previous installation of scapp (1 CPU limit per auth-service pod)
  - Metrics-server installed with metric-resolution to 5s

https://github.com/kubernetes-sigs/metrics-server
https://github.com/kubernetes-sigs/kind/issues/398#issuecomment-621143252

Security and network plugins

- Namespaces
  - A way to split logically a cluster in isolated groups
- Security and Role Based Access Control (RBAC)
  - Key based authentication outside of the cluster (for kubectl for instance)
  - API access control in and out of the cluster
  - Data encryption
  - Network control for pods
- CNI (Common Network Interface): Calico, Flannel, Cilium, …
  - SDN for container networking
  - Different performances and features such as logging, encryption, …
  - For basic usage, default is enough

https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/
https://kubernetes.io/docs/concepts/security/overview/

Kubernetes installations

- "Regular" installations
  - "Kubernetes the Hard Way"
  - Kubespray (Ansible based)
- Micro installations (limited features)
  - VM based (Minikube)
  - Docker based (Kind, microk8s)
  - Docker Desktop (Windows/Mac)
- Cloud
  - AKS (Azure), EKS (AWS), GKE (GCE), Alibaba..
- Edge computing (usage of small computing units near the users for better latency)
  - Kubeedge
  - K3S

- Kind installation (Linux) used for demos
  - For a minimal one node cluster installation, two commands:
    - Install https://kind.sigs.k8s.io/docs/user/quick-start/
    - kind create cluster
  - Possibility to set version of K8S master/node, mount host volumes, forward ports, etc.

Package manager : Helm

- Package manager
  - Definition, installation and upgrade of Kubernetes applications with “Helm charts”
  - Charts use enriched YAML files with variables, processing
- Features
  - Chart versioning
  - Powerful Go based templating engine
  - Hooks with pre/post processing
  - Charts composition (subcharts)
  - Packaging in tar.gz files
  - Usage of local or remote repositories
- Lifecycle
  - Replace variables and apply templating logic in YAML templates
  - Deploy subcharts
  - Deploy YAML templates
  - If set, trigger hooks before/after each object deployment / upgrade / suppression

https://helm.sh/docs/topics/charts/

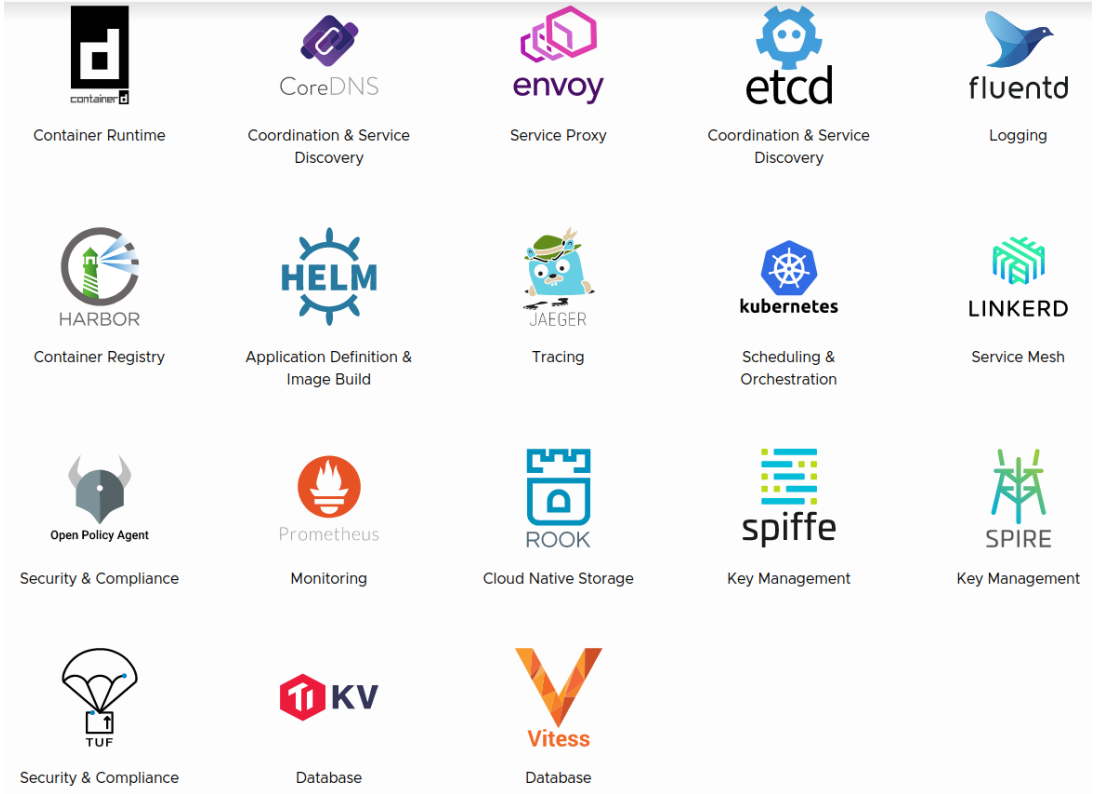
Example on a Helm chart: Wordpress

- With Helm installed
  - helm repo add bitnami https://charts.bitnami.com/bitnami
  - helm install my-release bitnami/wordpress

https://artifacthub.io/
https://artifacthub.io/packages/helm/bitnami/wordpress
https://github.com/helm/charts/tree/master/stable/wordpress

Extensions and tooling

- Too many modules !
- Cloud Native Computing Foundation (CNCF)
  - Cloud Native standardization association since 2015
  - Part of nonprofit Linux Foundation
  - « Vendor-neutral home for many of the fastest-growing open-source projects »
  - Also organizes events, certifications
- Projects split in categories
  - « Graduated » (see on the right)
  - « Incubating » (Istio, KNative, OpenTelemetry, KubeEdge ...)
  - « Sandbox » (chaos mesh, k3s ...)



https://www.cncf.io/projects/

Extensibility

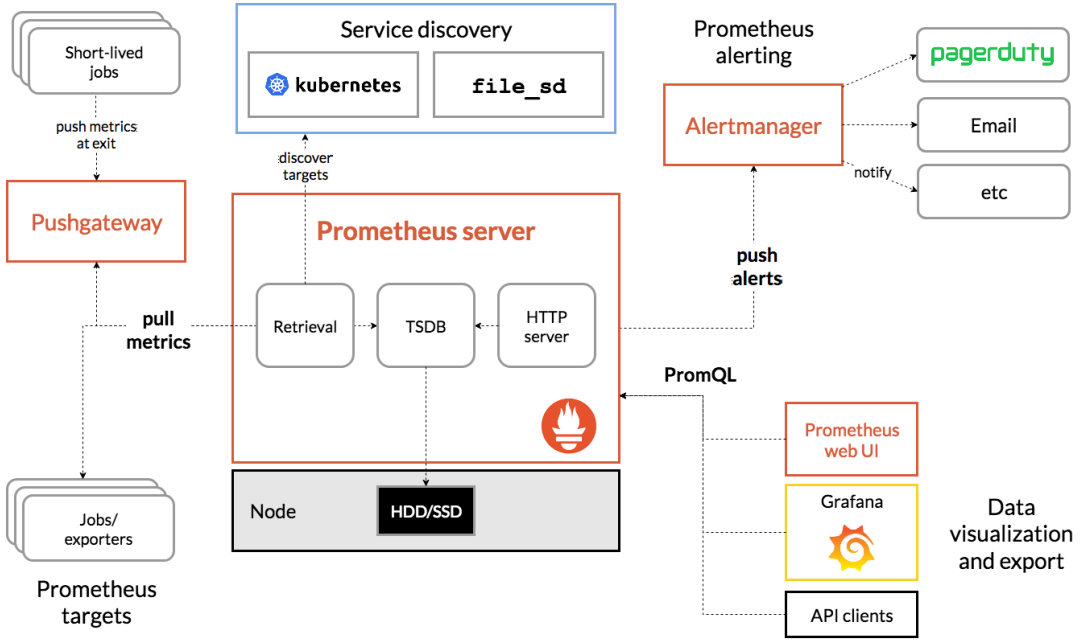
- Extensible normalized API
- YAML description files can be used for custom objects
- Custom Resources and Custom Resources Definition (CRD)
- Kubernetes is written in Go, lots of external modules as well

- Operators
  - Some maintenance tasks can be complicated
    - Dump database, jobs
  - Automation of these tasks with CRD
  - Example: MariaDB operator
  - Operators development
    - Lots of libraries, but can be complicated to prepare/debug compared to Helm charts (use code)
  - Interesting generic YAML based operator library: Kudo

https://operatorhub.io/

Monitoring : Prometheus

- Get telemetry measurements about applications in a centralized way
- « De-facto » standard often coupled with Grafana UI
  - Basis for OpenMetrics
- Principle : Prometheus « pulls » the metrics
  - Numeric metrics get at intervals
  - Prometheus pull metrics from inapp metrics Webserver
  - Adapters for applications not supporting it (ex: MySQL)
- Time series database + PromQL + Alerts
- Connectors, standard API in most modern applications
- Helm chart and operator … and usable with Docker too



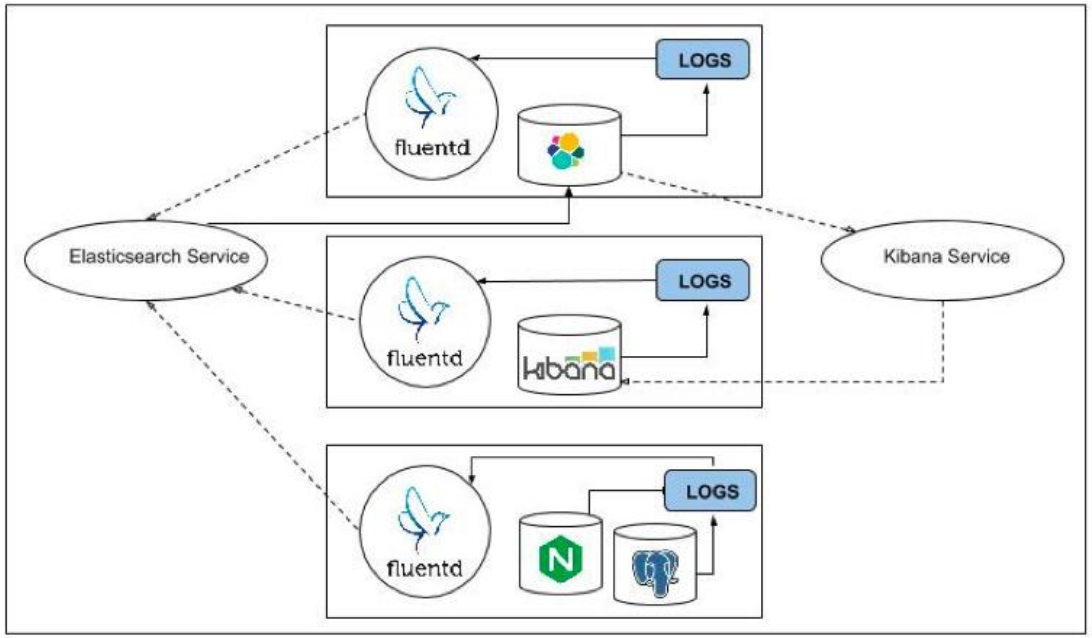
https://prometheus.io/docs/introduction/overview/

Demo Prometheus

- Install Prometheus with Helm (already done here)
- Launch Artillery for 2 minutes
- We check the results on Grafana/Prometheus

Centralized log management

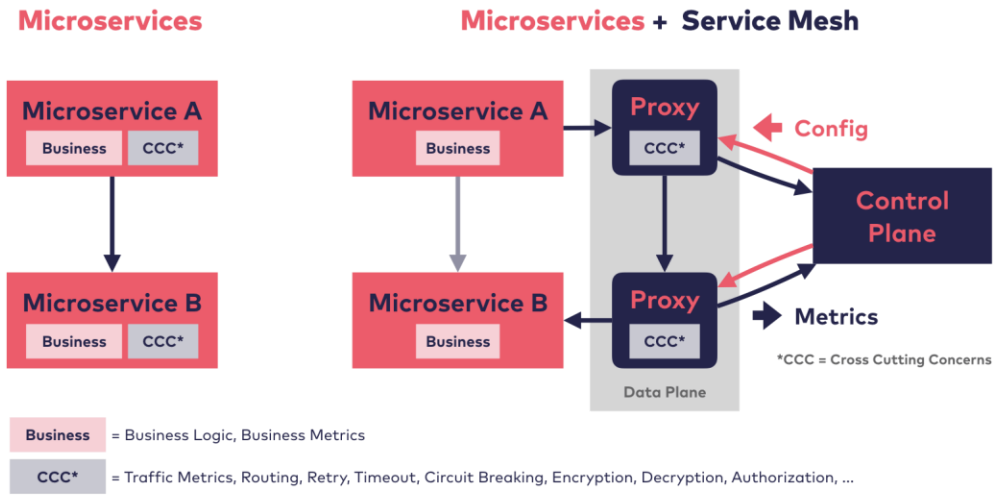
- As telemetry Logs can be complicated to manage with high number of containers
- Most used stack : EFK (also Loki)
  - Elasticsearch (db and search engine)
  - Daemonset Fluentd / Fluentbit (log router)
  - Kibana (UI)
- Lots of connectors
  - Docker/Kubernetes logs
  - Databases, syslog, AWS SQS …
- Helm charts and an operator



https://artifacthub.io/packages/helm/kiwigrid/fluentd-elasticsearch
https://operatorhub.io/operator/logging-operator

Service meshes

- Monitoring and adaptation of communications, useful in Microservices applications, without change in codebase
- Usually « Sidecar » container with a Layer 7 proxy added in pods that capture network trafic, and redirect to services/pods
- Features
  - Security (mTLS)
  - Resilience
  - Routing
  - Monitoring (distributed tracing)
- Dynamic configuration
- Tools
  - Istio, LinkerD …
  - Helm charts and operators



https://servicemesh.es/

Conclusion

- Comparison with Docker Swarm
  - Positive
    - Features for HA, elasticity, security, network …
    - Integration with Cloud (storage, load balancers)
    - Huge ecosystem and community
    - Extensibility
    - Lots of online resources
  - Negative
    - Steep learning curve
    - Complexity
      - Swarm : one compose file vs several for Kubernetes
      - Installation on an on-premises cluster can be complex
      - Installation and management of storage and access to cluster can be complicated on-premises
    - Overhead (hw resources and time)

- Used a lot in private companies so important to know about it
- Unavoidable today for medium/large scale container-based application deployments
- For small projects/testing, prefer Docker Compose / Docker Swarm

- Some Master Thesis / PhD Thesis on these subjects (large scale / k8s) so keep in touch with Pr. Rivière or me if you are interested in the future