Chmurowisko

www.chmurowisko.pl

Docker

Module Overview

Chmurowisko

- History of Docker
- Docker images
- Dockerfile
- Docker containers
- Docker image repositories

Why containers?

Chmurowisko

Before containers we needed to face some of these issues:

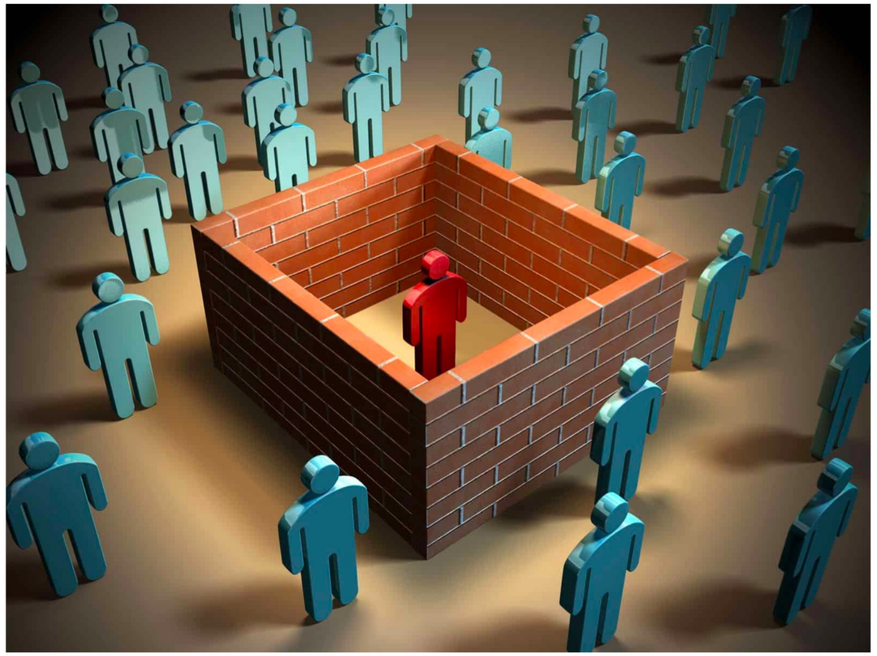
- Dependenecies between applications
- Dependencies from OS
- Dependencies from hardware
- Different deployment methods on different environments
- Lack of isolation (versioning conflicts, DLL hell)

Same application behaves differently depending on the environment, OS, hardware, etc.

How to resolve these issues?

Chmurowisko

We need isolation!



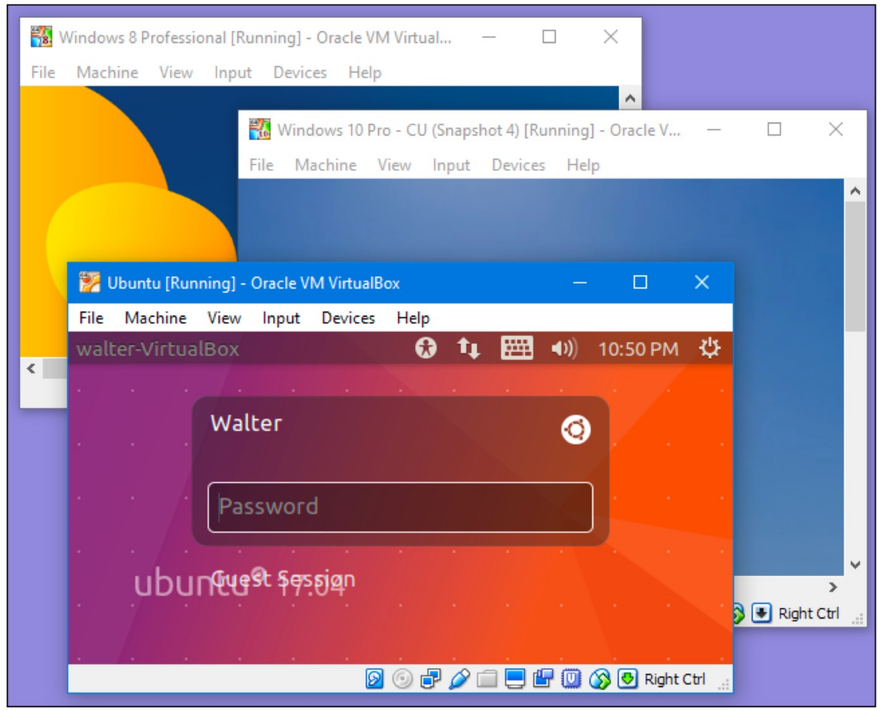
Benefits of isolation:

- You can run same application on different OS, hardware
- No versioning conflicts
- Same abstraction for running app on differents environments
- It doesn’t matter if you run the app in a cloud, in local datacenter or on a local machine

How can we achieve isolation?

Chmurowisko

Basically there are two ways to achieve isolation:

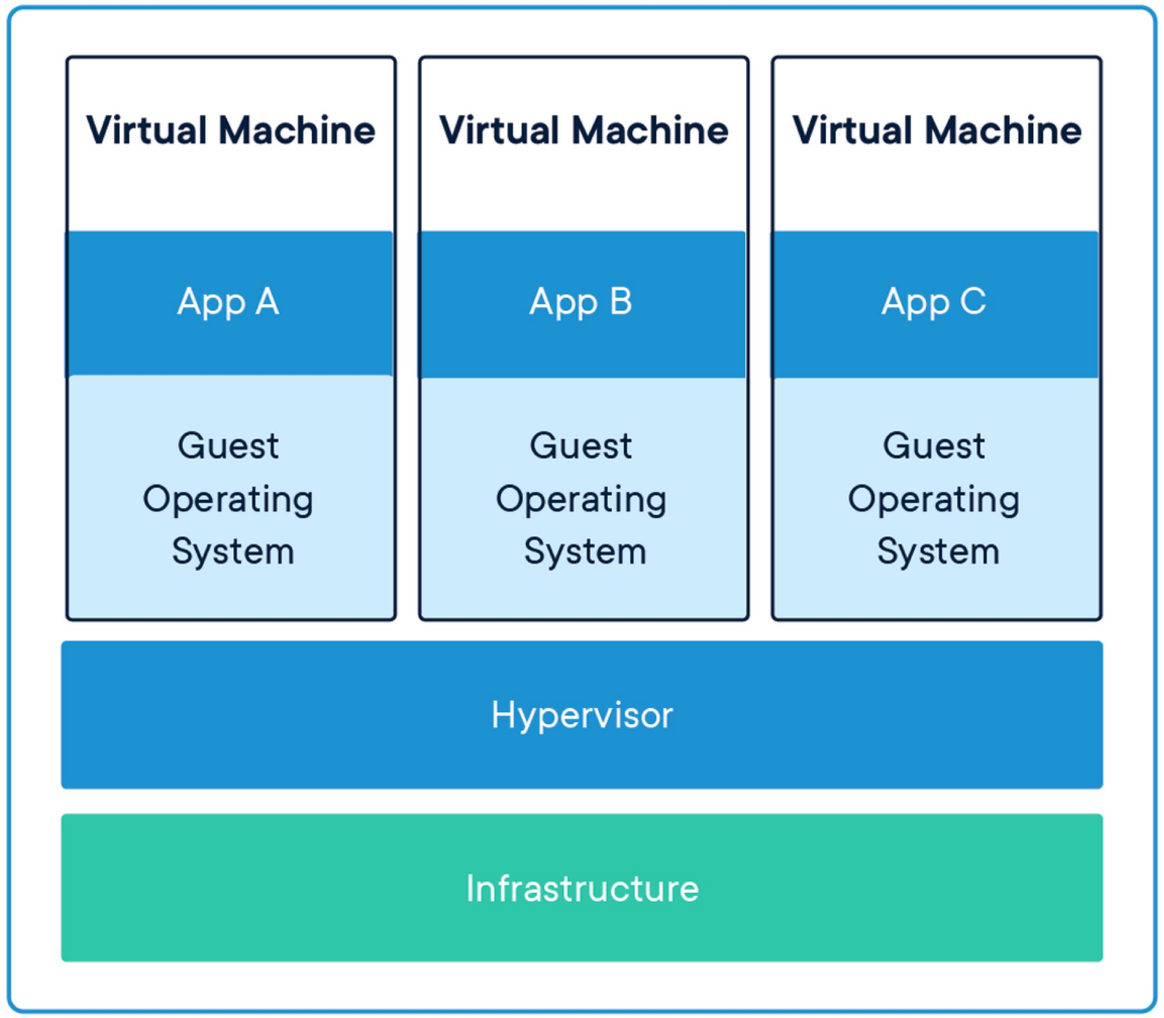


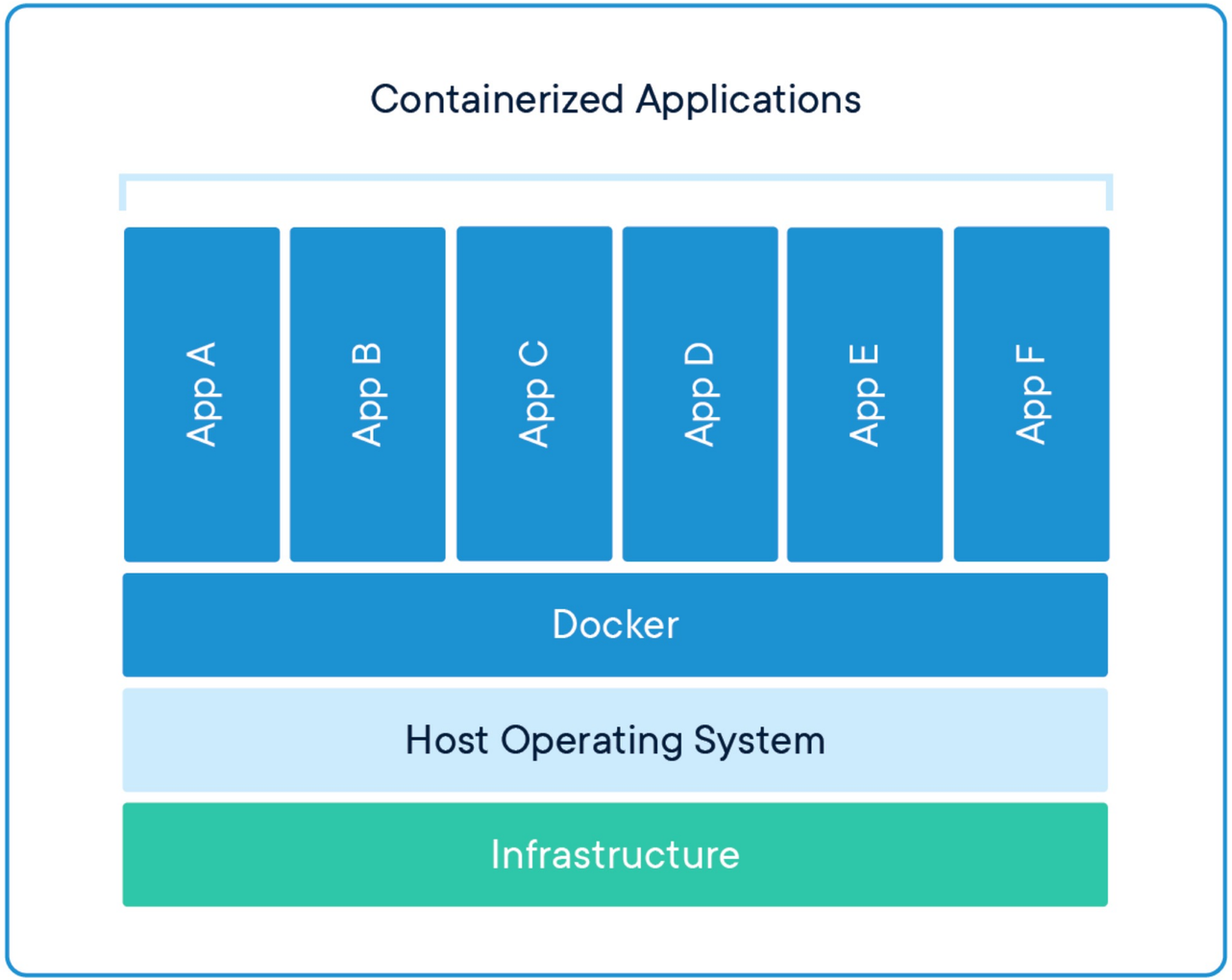
Virtual machines

Containers

Containers vs Virtual Machines

Chmurowisko





Advantages of Containerisation

Chmurowisko

- Forget about dependency nightmares
- Consistent progression from DEV -> TEST -> QA -> PROD
- Isolation - performance or stability issues of App A in container A, wont impact App B in container B.
- Better resource management.
- Extreme code portability
- Microservices

Containers vs Virtual Machines

Chmurowisko

Why Containers?

- Flexible: Even the most complex applications can be containerized.
- Lightweight: Containers leverage and share the host kernel.
- Interchangeable: You can deploy updates and upgrades on-the-fly.
- Portable: You can build locally, deploy to the cloud, and run anywhere.
- Scalable: You can increase and automatically distribute container replicas.
- Stackable: You can stack services vertically and on-the-fly.

What did exists before containers?

Chmurowisko

- Linux Chroot jail
- Linux namespaces
- Virtualization
- Containers

Containers in Linux existed much earlier than Docker came into being

Containers technologies

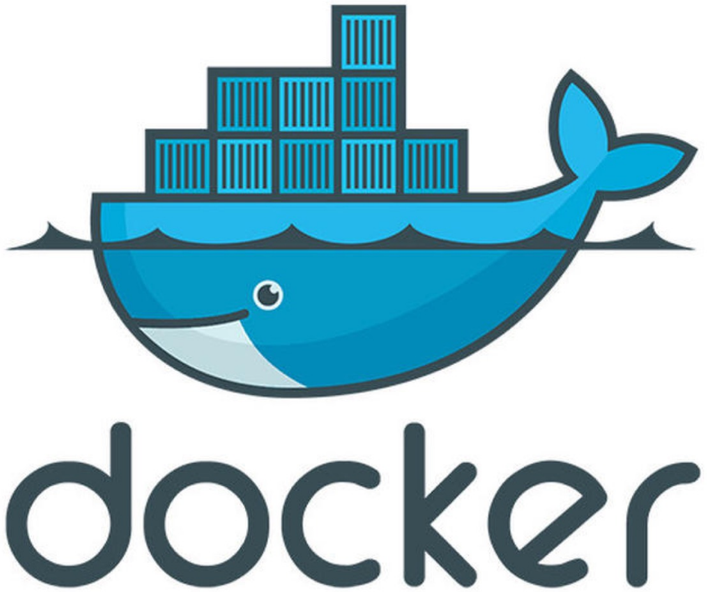
Chmurowisko

- OpenVZ - 2005
- LXC – 2008
- Docker – 2013
- Rkt - 2014
- LXD – 2015
- Windows Server Containers – 2016
- And many others…

Chmurowisko

Our Company

We can help drive your business into the global digital economy by providing the only container platform to provide the choice, agility and security needed to usher in the new era of application computing



Docker Editions

Chmurowisko

- Community Edition (CE) – free and open source
- Enterprise Edition (EE) - Premium version of Community Edition (3 pricing tiers) -
- Docker EE comes with additional features:
  - Certified Docker images
  - Advanced image and container management
  - Support from Docker Team
  - Docker Datacenter
  - Docker Security Scanning

Capabilities	Community Edition	Enterprise Edition Basic	Enterprise Edition Standard	Enterprise Edition Advanced
Container engine and built in orchestration, networking, security	✓	✓	✓	✓
Certified infrastructure, plugins and ISV containers		✓	✓	✓
Image management			✓	✓
Container app management			✓	✓
Image security scanning				✓

Docker on Linux

Chmurowisko

- Docker was using LXC (Linux Containers)
- Docker use the containerization features in the Linux kernel
- Docker use Linux kernel to setup containers
- Containers run ON Linux, not Docker
- Any distribution running version 3.10 (released in 2013) or greater of the Linux kernel
- CentOS, Debian, Fedora, Ubuntu, Red Hat and ….

Docker Desktop

Chmurowisko

- Application for MacOS and Windows
- Windows and Linux containers on Windows
- Enterprise and Community
- Docker Toolbox (for older Mac and Windows)

Feature	Docker Desktop (Community)	Docker Desktop Enterprise
Docker Engine	X	X
Certified Kubernetes	X	X
Docker Compose	X	X
CLI	X	X
Windows and Mac support	X	X
Version Selection		X
Application Designer		X
Custom application templates		X
Docker Assemble		X
Device management		X
Administrative control		X

Docker Desktop for Windows

Chmurowisko

- Available for free*
- Native Windows application
- Run on Windows 10 and on Windows Server 2016
- Requires Windows 10 (x64) Pro
- Requires Hyper-V

Docker Desktop for Mac

Chmurowisko

- Available for free*
- Native MacOS application
- Based on **xhyve**
- Only Linux containers
- Minimum OS version: MacOS Sierra 10.12
- Mac 2010 model or newer

Docker components

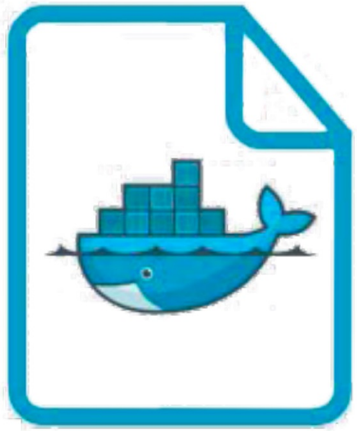
Chmurowisko

- Docker images
- Docker container
- Layers / Union file system
- DockerFile
- Docker Engine
- Docker Client
- Docker registries / Docker hub

Docker image

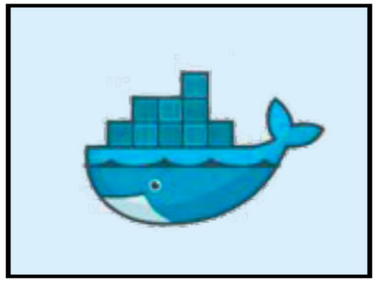
Chmurowisko

- Docker containers are based on Docker images.
- A Docker image is a binary that includes all of the requirements for running a single Docker container, as well as metadata describing its needs and capabilities.
- Docker containers only have access to resources defined in the image, unless you give the container additional access when creating it.
- Docker images are defined using Dockerfiles



Dockerfile

Build



Docker
Image

Run

Docker
Container

Docker CLI

Chmurowisko

- Command line interface for Docker
- Client for running Docker Engine to create images and containers
- Installed with Docker Engine
- https://github.com/docker/cli

Dockerfile

Chmurowisko

- Describe the way to build your container
- Imperative
- Instruction how to create your Image/Container
- Plain Text

Dockerfile

Chmurowisko

```
FROM ubuntu:14.04

RUN echo "Hello world" > /test/hello_world.txt

CMD ["cat", "/test/hello_world.txt"]
```

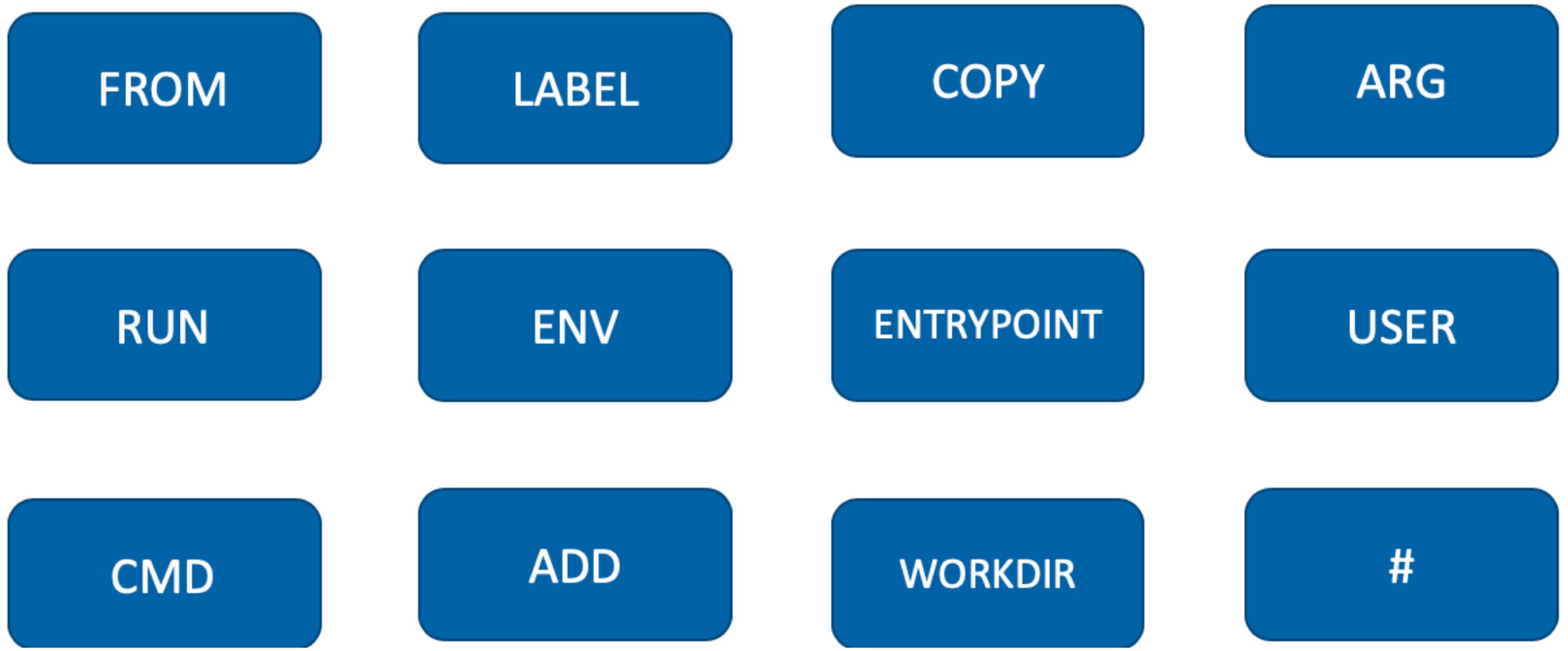
Dockerfile – best practice

Chmurowisko

- Reduce the dockerfile image layers
- Image build latency is affected by the specific dockerfile architecture attributes, especially, number of image layers, size of each layer, and the diversity of instructions
- Choose wisely your Base Image [FROM]
- Mulit Stage Dockerfile
- Create ephemeral containers
- Be aware of build context (use .dockerignore)
- Don’t install unnecessary packages
- Benefit from cache
- Label your images
- Read „Best practices for writing Dockerfiles”

Dockerfile commands

Chmurowisko



Docker build

Chmurowisko

- Build an image from a Dockerfile and a „context”
- „Context” is a set of files located in a specific PATH or URL

```
docker build . // build an image with Dockerfile and context in current directory
docker build –f Dockerfile.test // use Dockerfile.test instead
docker build –t myimage . // tag an image
docker build github.com/creack/docker-firefox // clone git repo and use it as a
context
```

Docker run

Chmurowisko

- Runs proces in isolated container
- Must specify an image
- Image can be local or from remote repository (DockerHub by default)
- Container can be run in detached mode (background) or in foreground mode

```
docker images // list all local images
docker run mylocalimage// run mylocalimage in foreground mode
docker run –d mylocalimage // run mylocalimage in background mode
docker run –it ubuntu // -t – allocate a pseudo tty, -i – keep STDIN open
docker run –rm myserver // remove container after it exits
docker run –p 80:80 myserver // map local port 80 to container’s port 80
docker run ubuntu whoami // run ubuntu image and execut „whoami” command
docker run –it myserver bash // runs container myserver and executes bash
```

Chmurowisko

www.chmurowisko.pl

Kubernetes

What problems K8s solves?

Chmurowisko

- Service discovery and load balancing.
- Horizontal scaling.
- Self healing.
- Automated rollouts and rollbacks.
- Secret and configuration management.
- Storage orchestration.
- Standarized way of deployment and operation

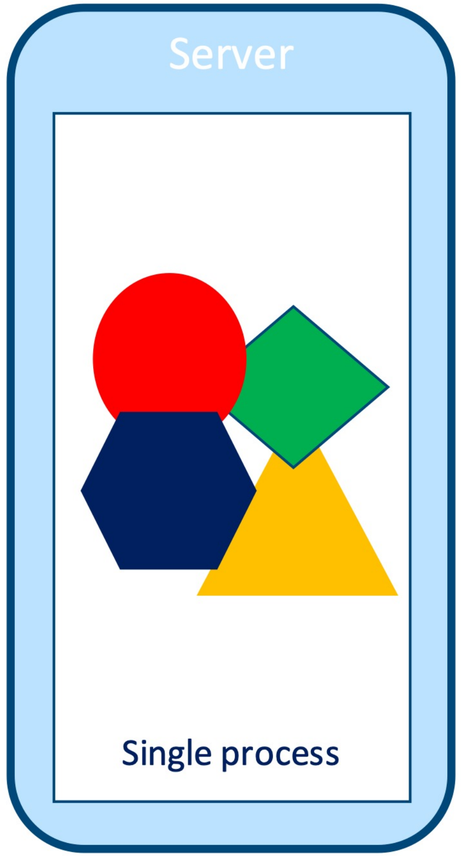
Chmurowisko

www.chmurowisko.pl

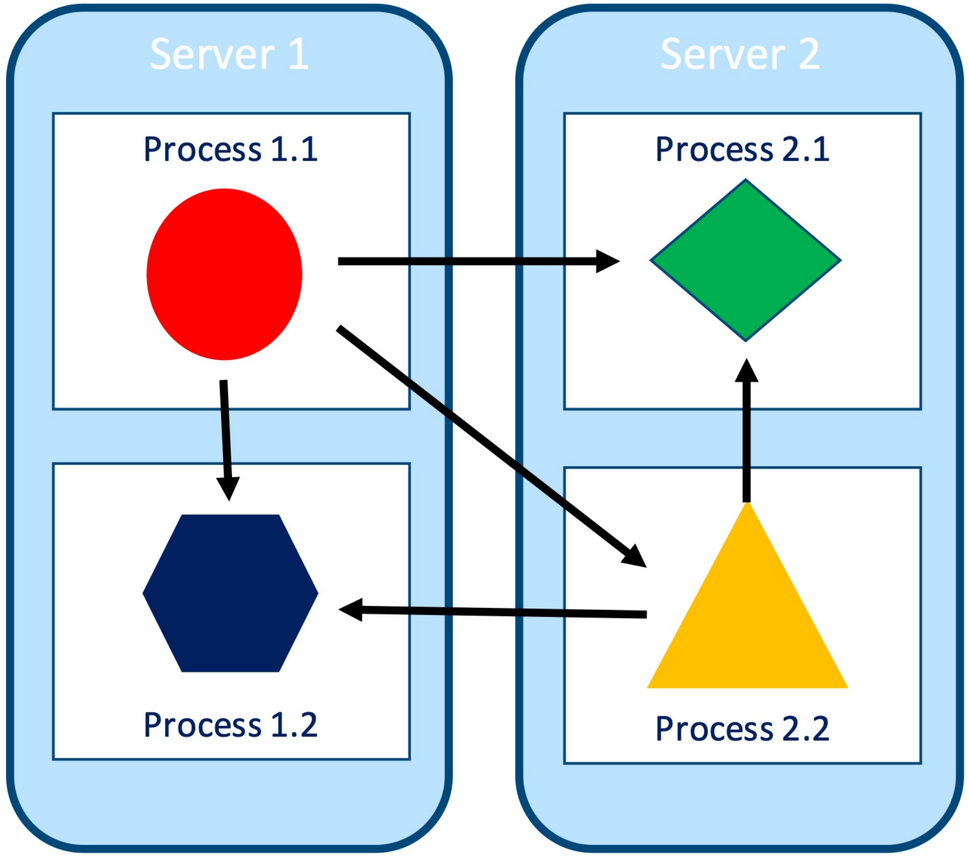
12 Factor Apps

Chmurowisko

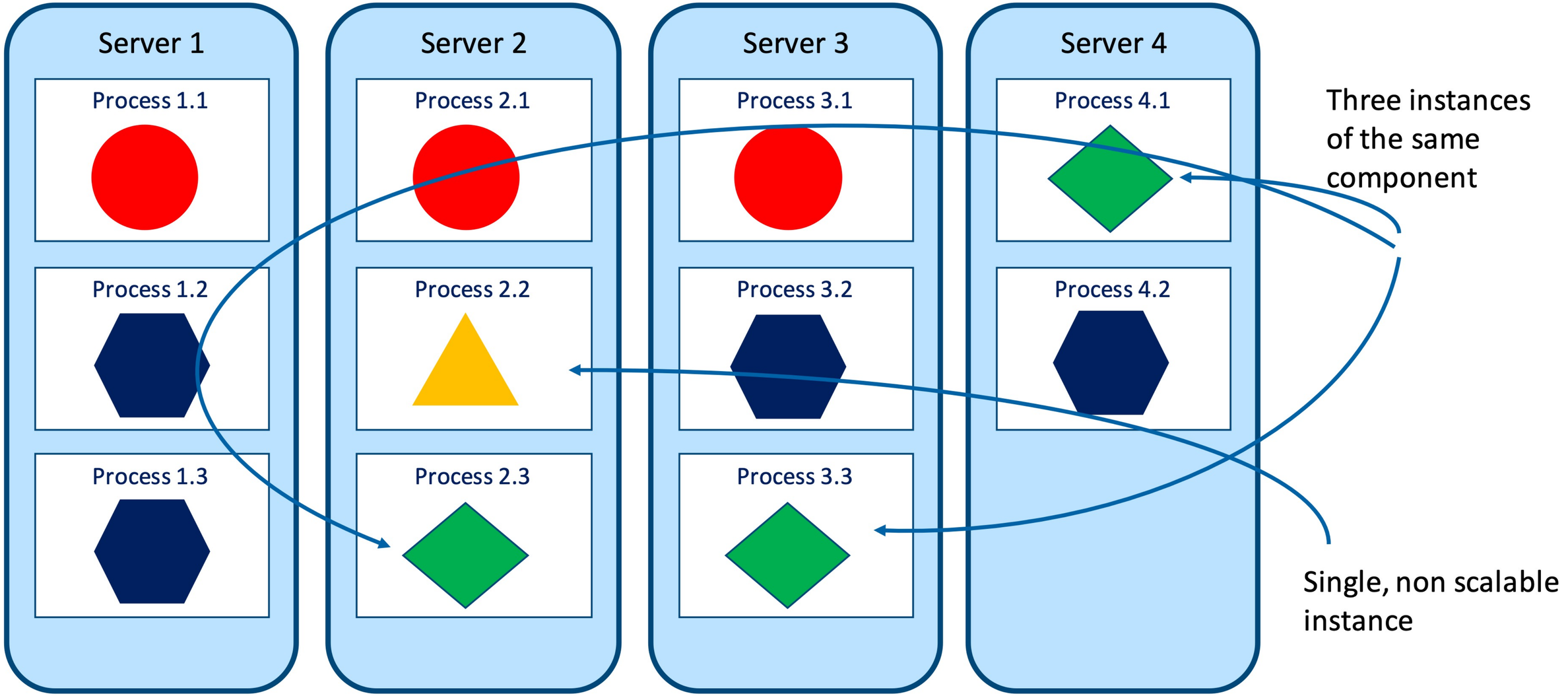
Monolith application



Microservices-based application



Chmurowisko



12 Factors

Chmurowisko

1. Codebase
2. Dependencies
3. Config
4. Backing services
5. Build, release, run
6. Processes
7. Port binding
8. Concurrency
9. Disposability
10. Dev/prod parity
11. Logs
12. Admin processes

1. Codebase

Chmurowisko

One codebase tracked in revision control, many deploys

Build on top of one codebase, fully tracked by a Version Control System (VCS). Deployments should be automatic, so everything can run in different environments without work. You should always have one repository for an individual application to ease CI/CD pipelines.

Code should not be shared between applications — you should not add dependencies of deployable services on each other. So, if there is common code you want to use across applications, create a repo for that code as a library

2. Dependencies

Chmurowisko

Explicitly declare and isolate dependencies

Do not copy any dependencies to the project codebase, instead use a package manager. Always remember to use the correct versions of dependencies so that all environments are in sync and reproduce the same behawior

- maven
- nuget
- npm

3. Config

Chmurowisko

Store config in the environment

Store the config in Environment Variable. There should be a strict separation between config and code. The code should remain the same irrespective of where the application is being deployed, but configurations can vary.

Use cases:

- Database connection properties
- Backing services credentials and connection information
- Application environment specific information such as Host IP, Port

4. Backing services

Chmurowisko

Treat backing services as attached resources

Treat backing services as attached resources as your services should be easily interchangeable. You must be able to easily swap the backing service from one provider to another without code changes. This will ensure good portability and help maintain your system.

Database, Message Brokers, other API-accessible consumer services such as Authorization Service, Twitter, GitHub etc., are loosely coupled with the application and treat them as resource.

5. Build, release, run

Chmurowisko

Strictly separate build and run stages

A twelve-factor application requires a strict separation between Build, Release and Run stages. Every release should always have a unique release ID and releases should allow rollback. Automation and maintaining the system should be as easy as possible. Then you put everything together in something that can be released and installed in the environment and then be able to run it.

- The Build phase takes code from VCS and builds an executable bundle
- In Release stage, an executable build is combined with environment specific configurations, assigned a unique release number, and made ready to execute on the environment
- In Release phate, package is executed on an environment using the necessary execution commands

6. Processes

Chmurowisko

Execute the app as one or more stateless processes

You should not be introducing state into your services, applications should execute as a single, stateless process. The Twelve-factor processes are stateless and share-nothing.

- Use a database to store state if needed for subsequent requests
- Avoid using sticky-session, and instead use scalable cache stores such as Memcached or Redis for storing session information

7. Port binding

Chmurowisko

Export services via port binding

Your service should be visible to others via port binding. If you built a service, make sure that other services can treat this as a resource if they wish

8. Concurrency

Chmurowisko

Scale out via the process model

Break your app into much smaller pieces rather than trying to make your application larger (by running a single instance on the most powerful machine available). Small, defined apps allow scaling out as needed to handle the varying loads. Each process should be individually scaled, with Factor 6 (Stateless), it is easy to scale the service

Don’t rely too much on threads in an application as vertical scaling can be limited for process running on server

9. Disposability

Chmurowisko

Maximize robustness with fast startup and graceful shutdown

Processes should be less time-consuming. Make sure you can run and stop fast. And that you can handle failure. Without this, automatic scaling and ease of deployment, development- are being diminished.

Run and Stop should be minimal to avoid catastrophic failures between different applications working as backing services

10. Dev/prod parity

Chmurowisko

Keep development, staging, and production as similar as possible

Keep development, staging, and production as similar as possible so anyone can understand it and release it. Continuous deployment needs continuous integration based on matching environments to limit deviation and errors

11. Logs

Chmurowisko

Treat logs as event streams

Treat logs as event streams. Logging is important for debugging and checking up on the general health of your application. At the same time, your application shouldn’t concern itself with the storage of this information. Instead, these logs should be treated as a continuous stream that is captured and stored by a separate service

12. Admin processes

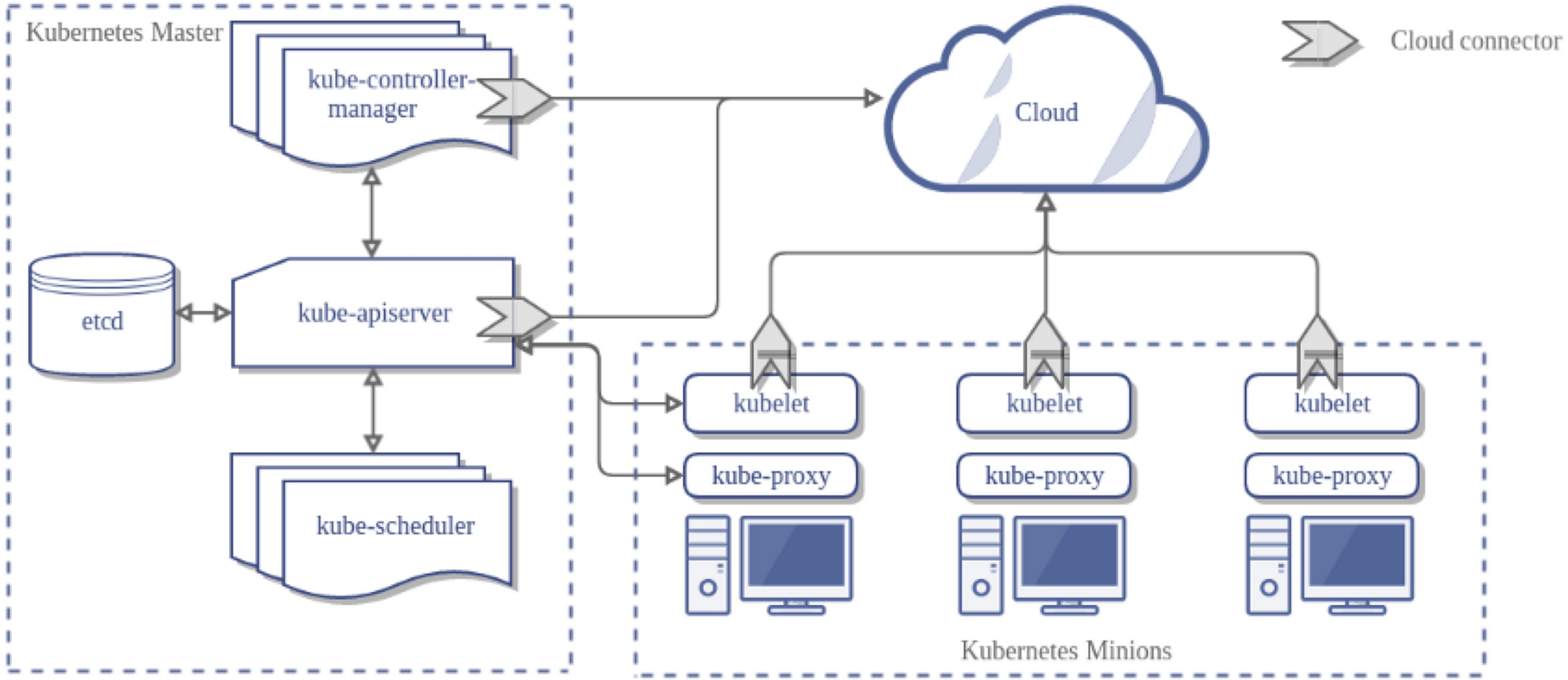
Chmurowisko

Run admin/management tasks as one-off processes

Run admin/management tasks as one-off processes — tasks like database migration or executing one-off scripts in the environment. To avoid messing with the database, use the tooling you built alongside your app to go and check the database

Basic K8s Architecture

Chmurowisko



Objects

Pod

Chmurowisko

- Basic unit in Kubernetes
- Consists of one or more containers. These containers share network namespace and IPC namespace
- Data on pods is volatile – use volumes
- Resides on one node
- Each pod has unique IP
- They are immutable

Pod

Chmurowisko

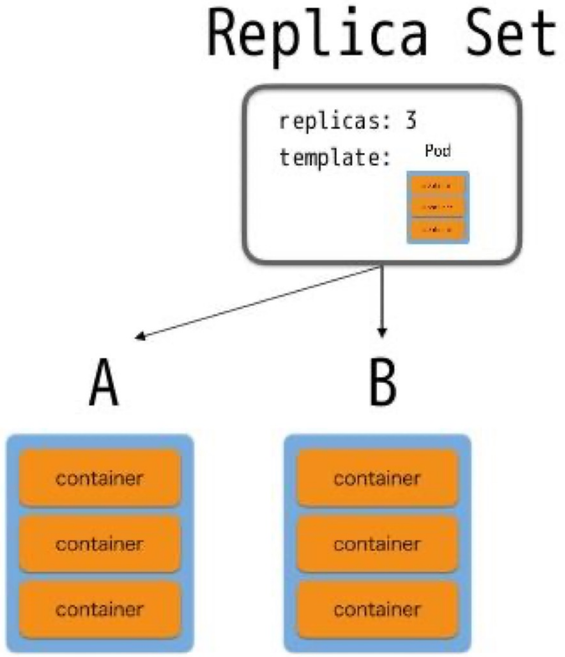
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: app1
spec:
  containers:
  - name: nginxname
    image: nginx
    ports:
      - containerPort: 80
```

ReplicaSet

Chmurowisko

ReplicaSet acts as a cluster-wide Pod manager, ensuring that the right types and number of Pods are running at all times.

Pods managed by ReplicaSets are automatically rescheduled under certain failure conditions such as node failures and network partitions.



ReplicaSet - Scaling

Chmurowisko

A ReplicaSet can be scaled up or down by updating the .spec.replicas field. The ReplicaSet controller ensures that a desired number of Pods with a matching label selector are available and operational.

- Imperative scaling:

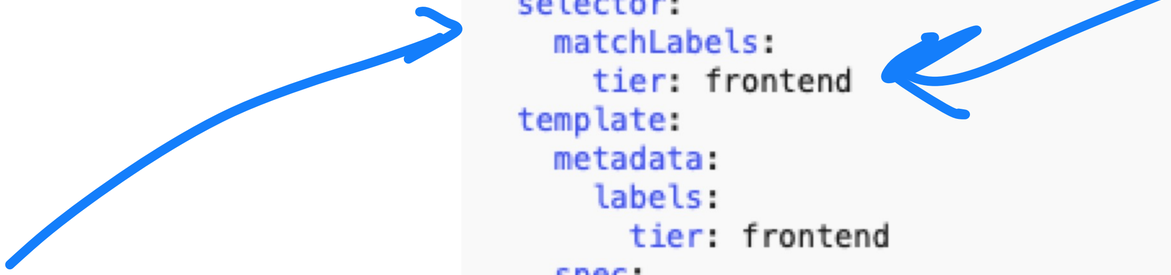
```
$ kubectl scale replicasets kuard --replicas=4
```

- Declarative scaling:

```
... spec:
replicas: 3
...
```

ReplicaSet

Chmurowisko



```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

ReplicaSet

Chmurowisko

Notes:

There are three reasones why you want to build a Replica Set:

- Redundancy
- Scale
- Sharding the service you want to offer.

The reconciliation loop is constantly running, observing the current state of the world and taking action to try to make the observed state match the desired state. To goal is to make the current state match the desired state.

ReplicaSet is decoupled from Pod – you can create the pod and then create Replica Set to manage this pod. On the other hand you may disconnect the Pod from ReplicaSet.

Deployment

Chmurowisko

Deployment helps to manage the release of the app, new version of app.

Move from one version to another by using rollout proces which can be managed.

Notes:

Deployment is responsible for deploying whole app – if this requires couple various RS or Service this is fine

Two deployment strategies:

-> Recreate

-> Rollingupdate

maxUnavailable – number of % - how many pods can be unavailable during the update (Recreate use 100%)

maxSurge

Deployment

Chmurowisko

- **spec.replicas:** The number of replica pods
- **spec.template:** A template pod descriptor which defines the pods which will be created
- **spec.selector** The deployment will manage all pods whose labels match this selector. When creat- ing a deployment, make sure the selector matches the pods on the template!

```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:1.7.9
        ports:
        - containerPort: 80
```

Rollout

Chmurowisko

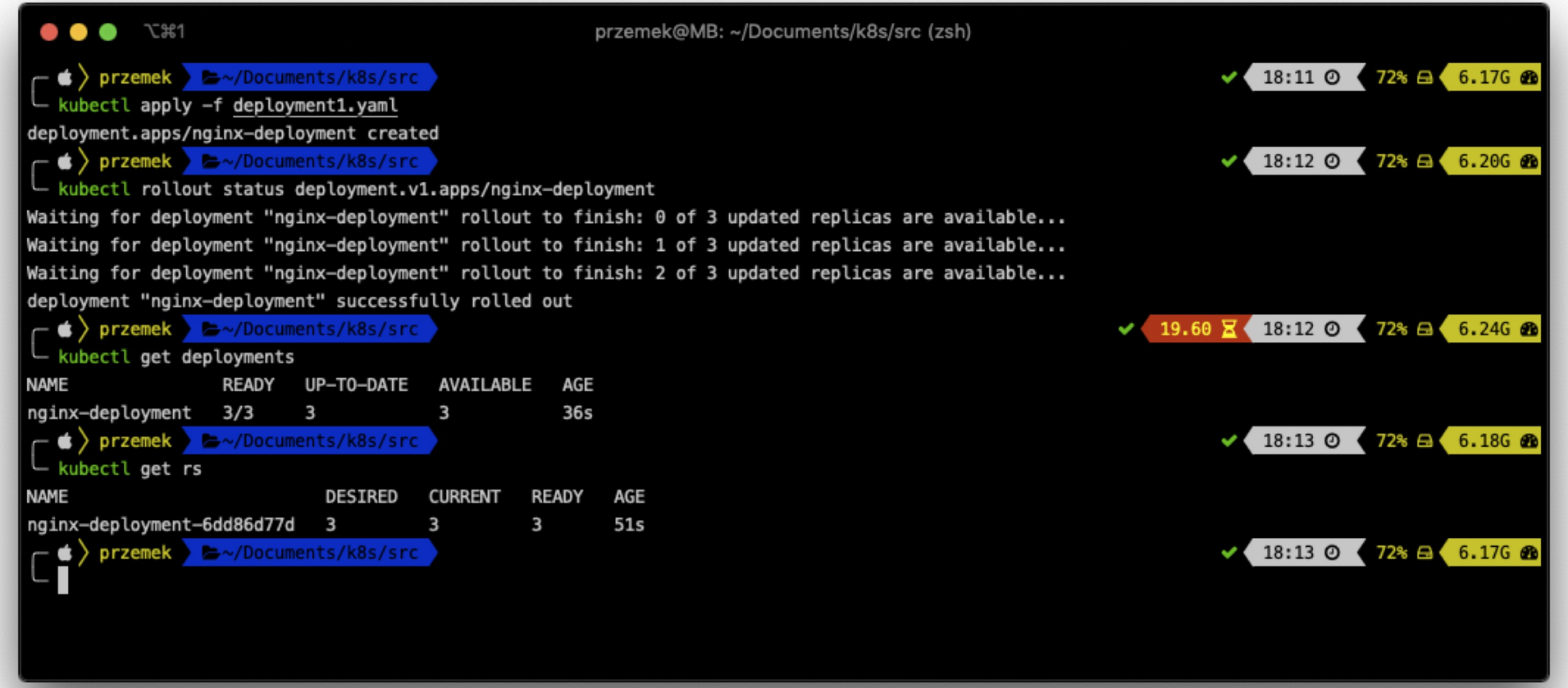
Deployment creates a ReplicaSet of pods.

- A Deployment named nginx-deployment is created.
- The Deployment creates three replicated Pods.
- The selector field defines how the Deployment finds which Pods to manage. In this case, you simply select a label that is defined in the Pod template (app: nginx).

```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:1.7.9
        ports:
        - containerPort: 80
```

Rollout

Chmurowisko



Rollout - Pod-template-hash Label

Chmurowisko

- The pod-template-hash label is added by the Deployment controller to every ReplicaSet that a Deployment creates or adopts
- This label ensures that child ReplicaSets of a Deployment do not overlap

Do not change this label !!!

Rollout – Deployment Strategies

Chmurowisko

- Recreate Deployment

All existing Pods are killed before new ones are created when

`.spec.strategy.type==Recreate`

- Rolling Update Deployment

Updates one pod at a time, rather than taking down the entire service at the same time

`.spec.strategy.type==RollingUpdate`

Rollout – Rolling Update

Chmurowisko

- Max Unavailable

`.spec.strategy.rollingUpdate.maxUnavailable`

Specifies the maximum number of Pods that can be unavailable during the update process. The value can be an absolute number (for example, 5) or a percentage of desired Pods (for example, 10%). The absolute number is calculated from percentage by rounding down. The default value is 25%.

- Max Surge

`.spec.strategy.rollingUpdate.maxSurge`

Specifies the maximum number of Pods that can be created over the desired number of Pods. The value can be an absolute number (for example, 5) or a percentage of desired Pods (for example, 10%). The absolute number is calculated from the percentage by rounding up. The default value is 25%.

Rollout – Updating a Deployment

Chmurowisko

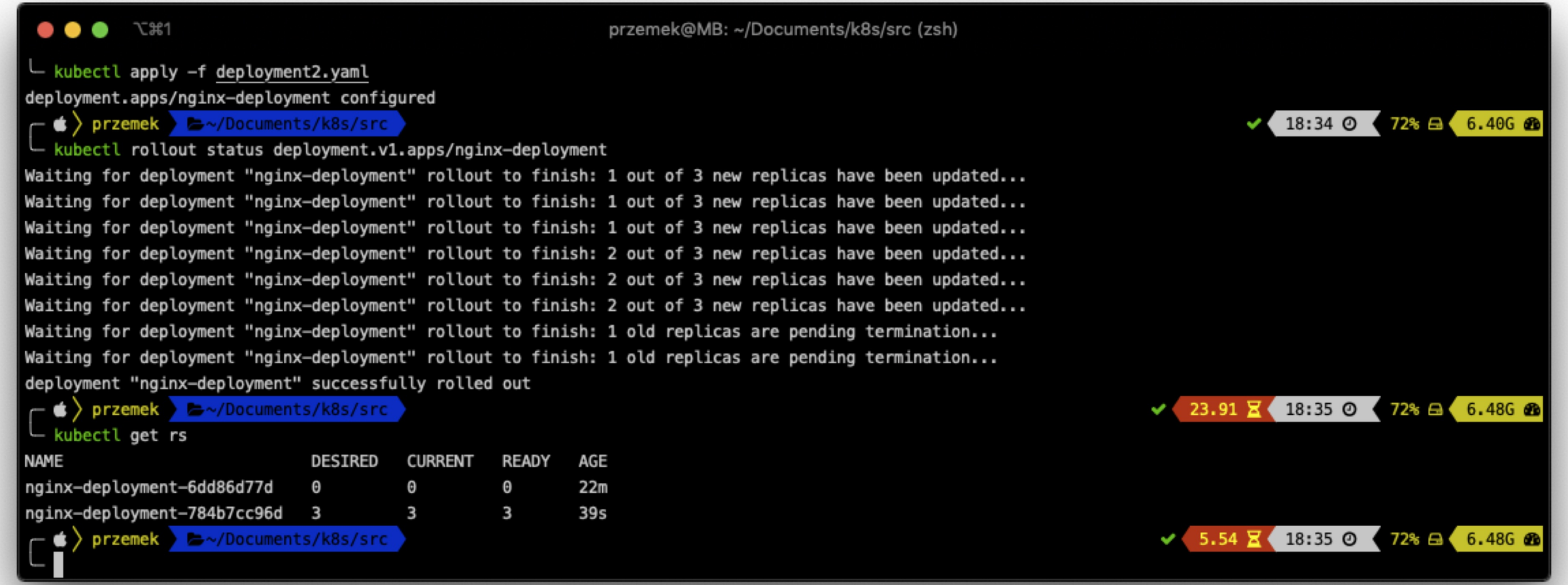
Note: A Deployment's rollout is triggered if and only if the Deployment's Pod template (that is, .spec.template) is changed, for example if the labels or container images of the template are updated. Other updates, such as scaling the Deployment, do not trigger a rollout.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
  annotations:
    kubernetes.io/change-cause: "Image update"
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.9.1
        ports:
        - containerPort: 80
```

Rollout – Updating a Deployment

Chmurowisko

Both old and new ReplicaSets managed by the deployment along with the images being used. Both the old and new ReplicaSets are kept around in case you want to roll back.

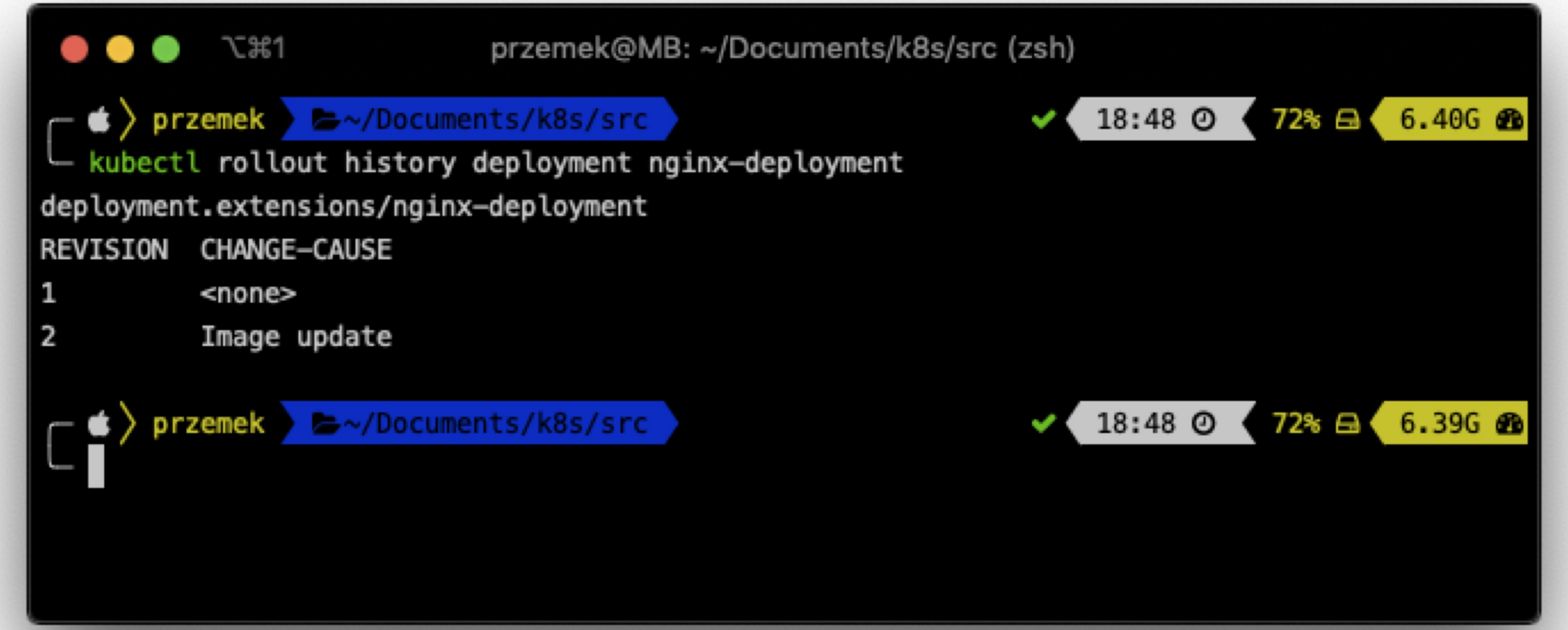


Rollout - History

Chmurowisko

Kubernetes deployments maintain a history of rollouts, which can be useful both for understanding the previous state of the deployment and to roll back to a specific version.

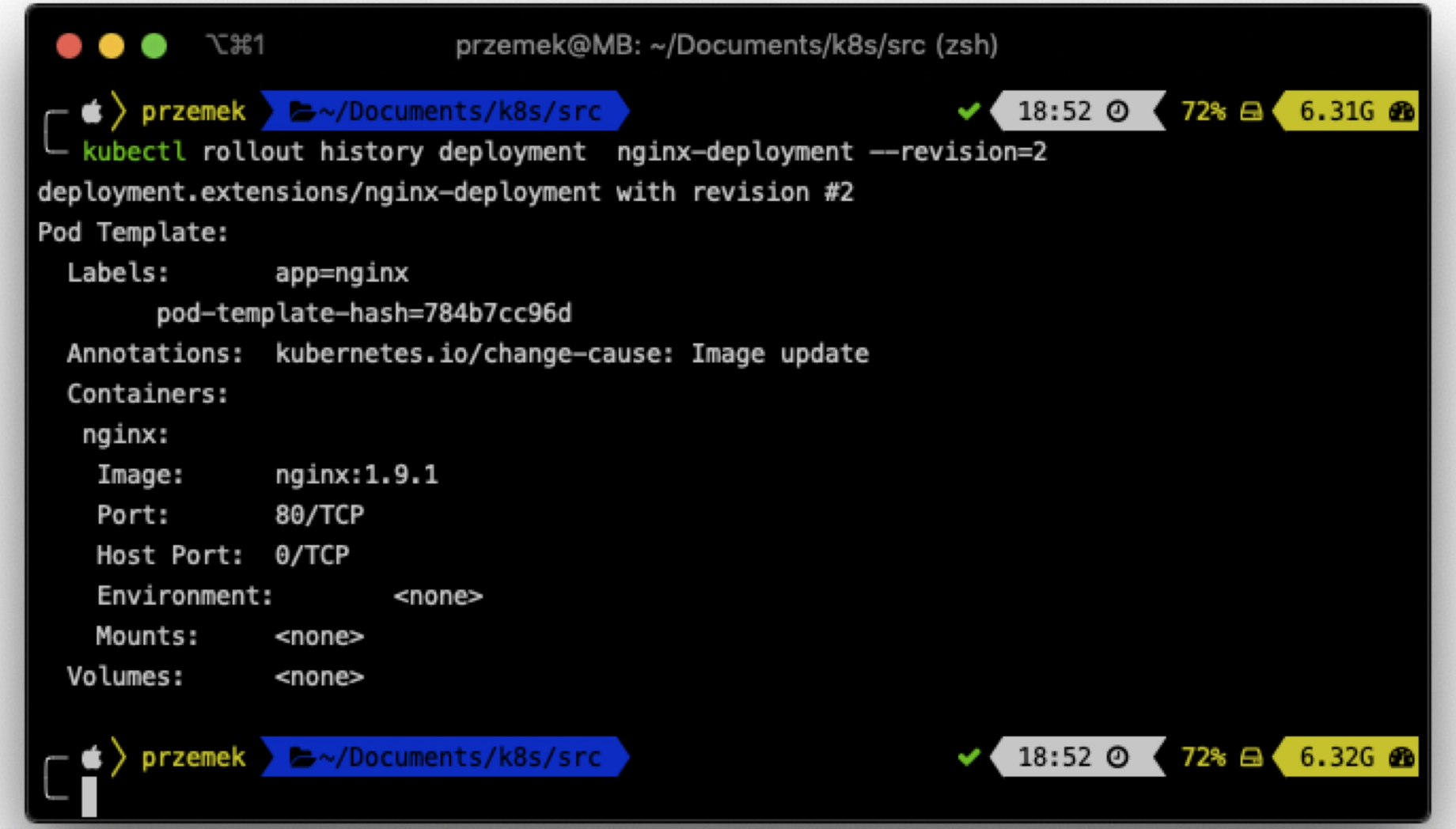
```
$ kubectl rollout history deployment nginx-deployment
```



Rollout - History

Chmurowisko

```
$ kubectl rollout history deployment  nginx-deployment --revision=2
```



Rollout - Undo

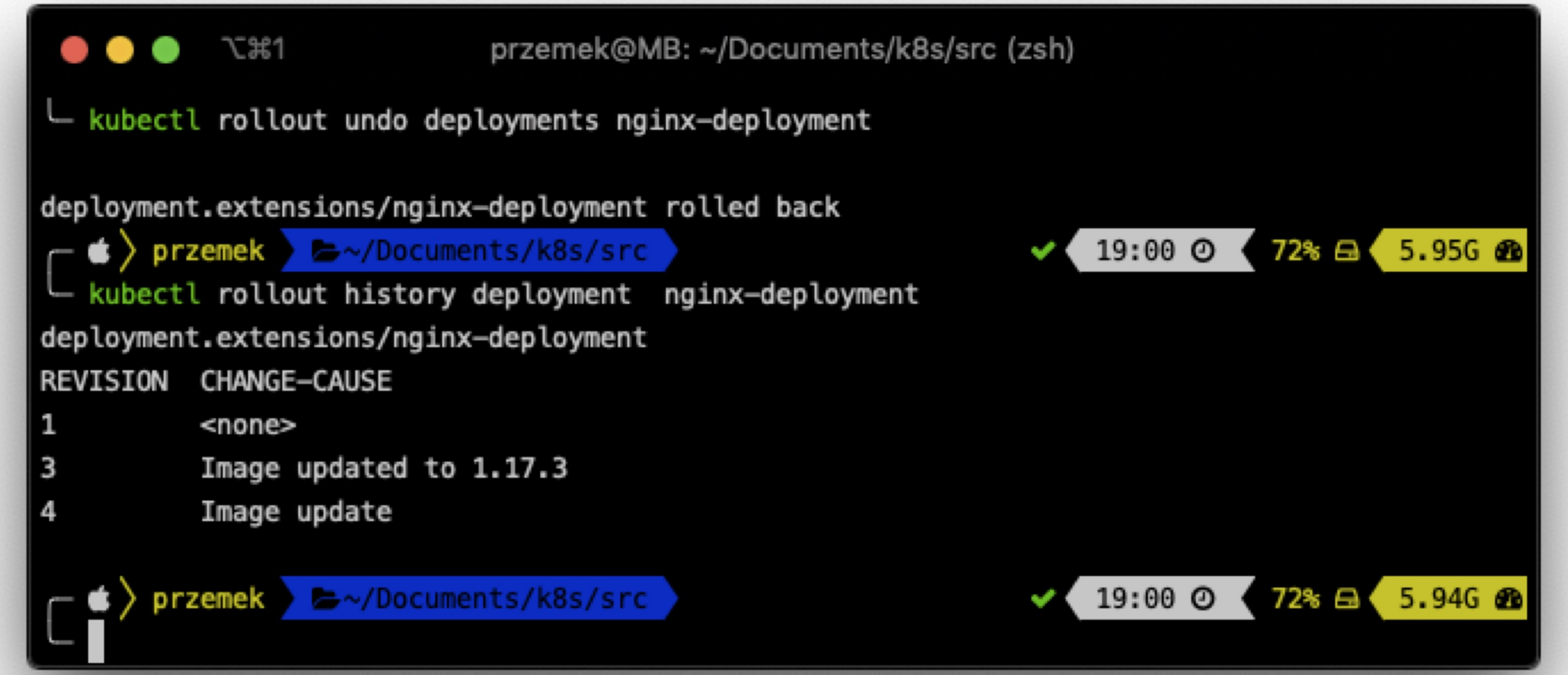
Chmurowisko

By default, all of the Deployment’s rollout history is kept in the system so that you can rollback anytime you want (you can change that by modifying revision history limit).

Rollout - Undo

Chmurowisko

```
$ kubectl rollout undo deployments nginx-deployment
```



Service

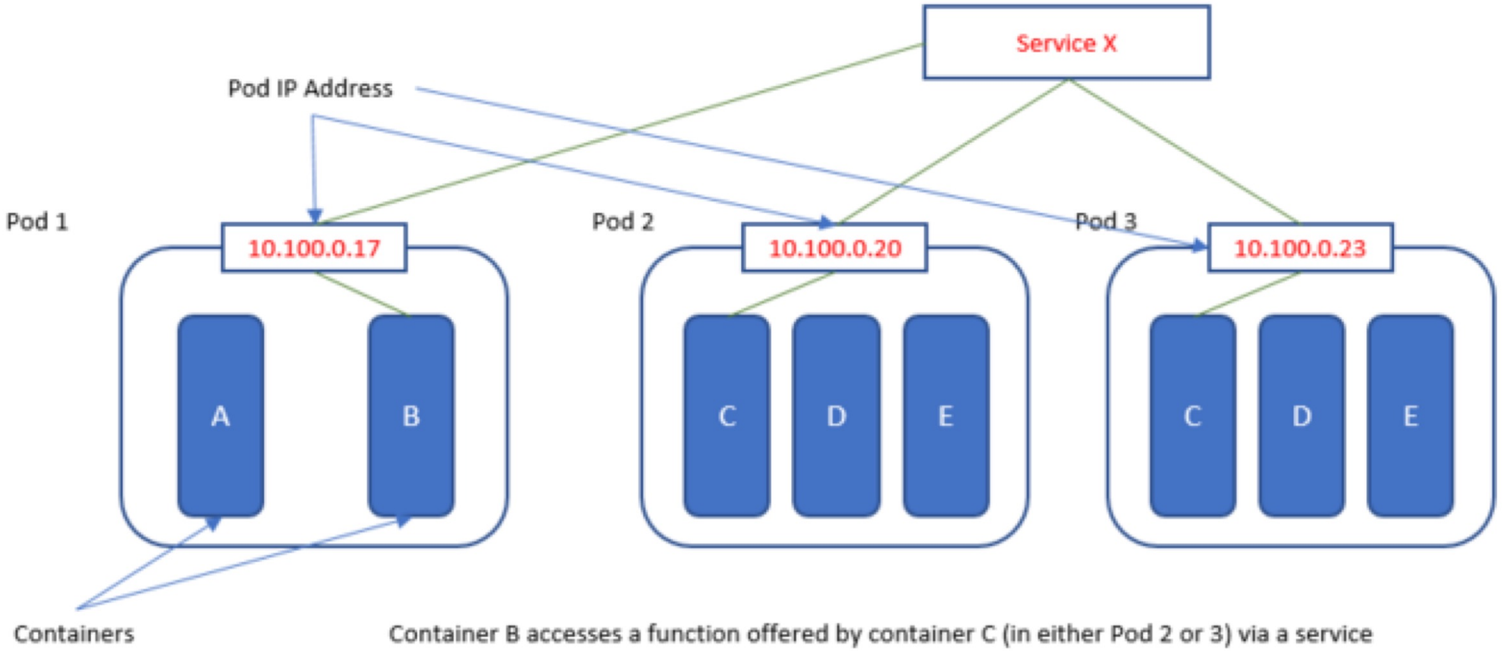
Chmurowisko

- An abstract way to expose an application running on a set of Pods as a network service.
- A Service object is a way to create a named label selector.
- Defines a logical set of Pods and a policy by which to access them
- Each service is automatically added to Endspoints object

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Service

Chmurowisko



Service - Types

Chmurowisko

- *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. You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>.
- *LoadBalancer* - Exposes the Service externally using a cloud provider's load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created.
- *ExternalName* - Maps the Service to the contents of the external Name field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up.

Service

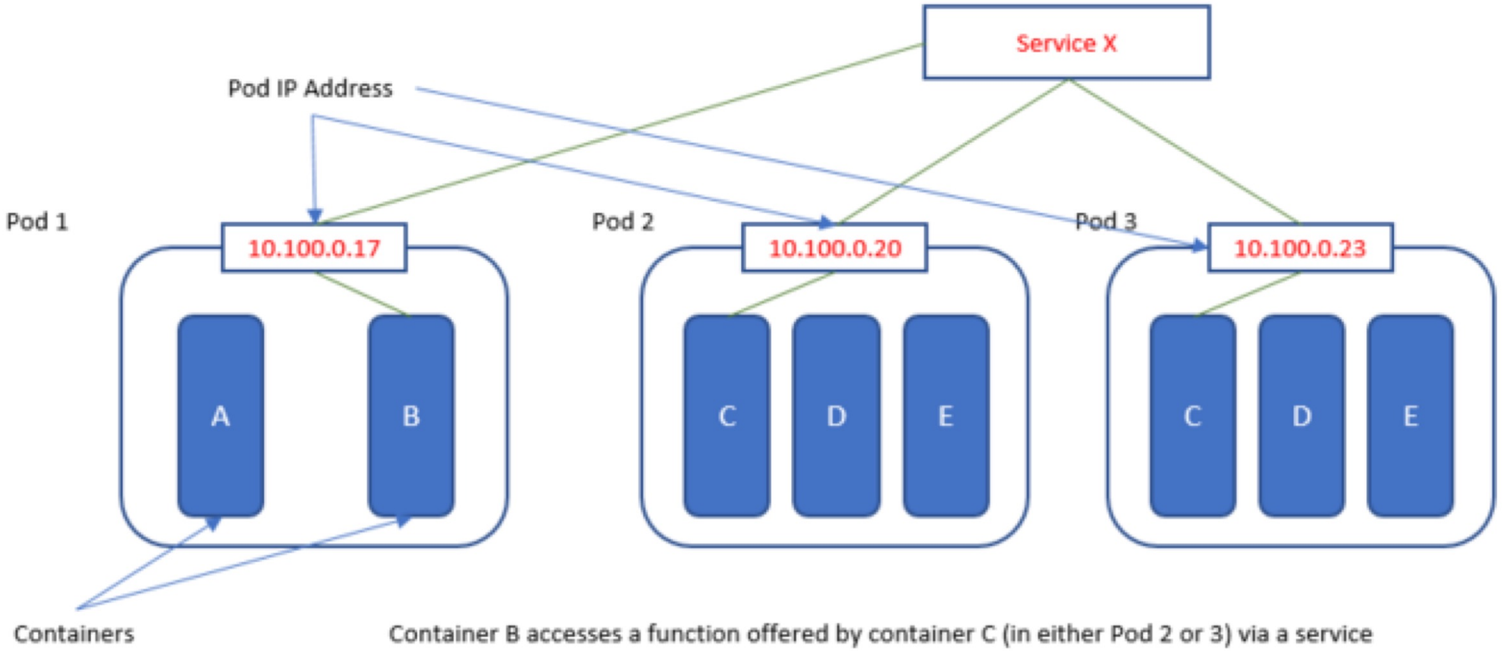
Chmurowisko

- An abstract way to expose an application running on a set of Pods as a network service.
- A Service object is a way to create a named label selector.
- Defines a logical set of Pods and a policy by which to access them
- Each service is automatically added to Endspoints object

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Service

Chmurowisko



Service – Without Selector

Chmurowisko

A service define a way to access pod, however:

- You want to have an external database cluster in production, but in test you use your own databases.
- You want to point your service to a service in another namespace or on another cluster.
- You are migrating your workload to Kubernetes and some of your backends run outside of Kubernetes.

```yaml
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

```yaml
kind: Endpoints
apiVersion: v1
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 1.2.3.4
    ports:
      - port: 9376
```

Service - Types

Chmurowisko

- *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. You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>.
- *LoadBalancer* - Exposes the Service externally using a cloud provider's load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created.
- *ExternalName* - Maps the Service to the contents of the external Name field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up.

Service Discovery

Chmurowisko

There are two ways to find a service:

Environment variables:

- **{SVCNAME}_SERVICE_HOST**
- **{SVCNAME}_SERVICE_PORT**

DNS

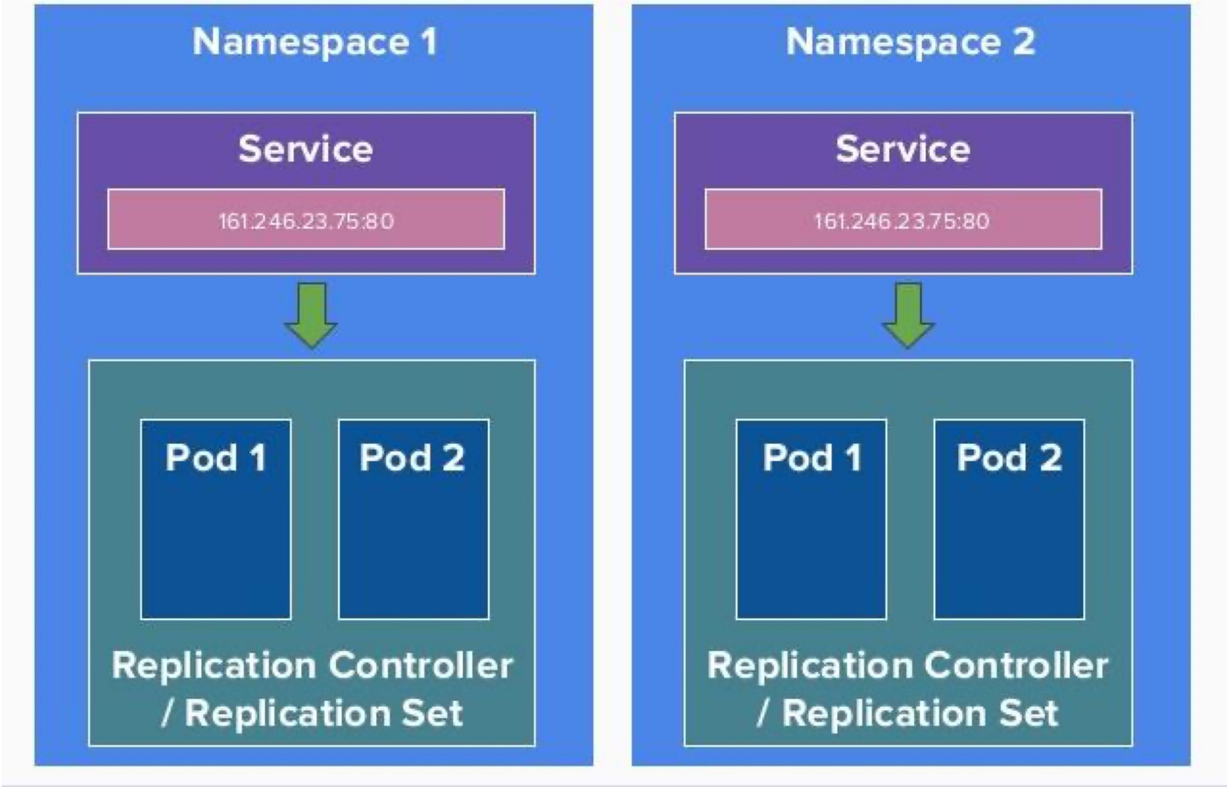
- **Cluster add-on (CoreDNS)**
- **Format: "my-service.my-ns"**

```
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

Namespace

Chmurowisko

- A virtual cluster backed by the same physical cluster
- A scope for names - names of resources need to be unique within a namespace
- A way to divide cluster resources between multiple users (you can use resource quotas)
- An object that groups other objects
- It is used to implement multi-tenancy on single cluster



Namespace

Chmurowisko

Three initial namespaces:

- *default* – default namespace for all objects with no other namespace
- *kube-public* – readable for all users, even unauthenticated
- *kube-system* – Kubernetes system objects

Namespace - DNS

Chmurowisko

Services create corresponding DNS entries:

<service-name>.<namespace-name>.svc.cluster.local

- If service just uses *service-name*, it will resolve to the service which is local to the current namespace
- Fully qualified domain name must be used to reach a service in a different namespace

Useful for using the same configuration across multiple namespaces.

ConfigMaps

Chmurowisko

ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.

- Object that defines a small file- system
- Set of variables that can be used when defining the envi- ronment or command line for your containers

ConfigMap is combined with the Pod right before it is run. This means that the container image and the Pod definition itself can be reused across many apps by just changing the ConfigMap that is used.

ConfigMaps

Chmurowisko

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2017-12-27T18:38:34Z
  name: config-multi-env-files
  namespace: default
data:
  color: purple
  how: fairlyNice
  textmode: "true"
```

ConfigMaps – Env Variables

Chmurowisko

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        # Define the environment variable
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              # The ConfigMap containing the value you want to assign to SPECIAL_LEVEL_KEY
              name: special-config
              # Specify the key associated with the value
              key: special.how
  restartPolicy: Never
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
        - name: LOG_LEVEL
          valueFrom:
            configMapKeyRef:
              name: env-config
              key: log_level
  restartPolicy: Never
```

ConfigMaps - Volumes

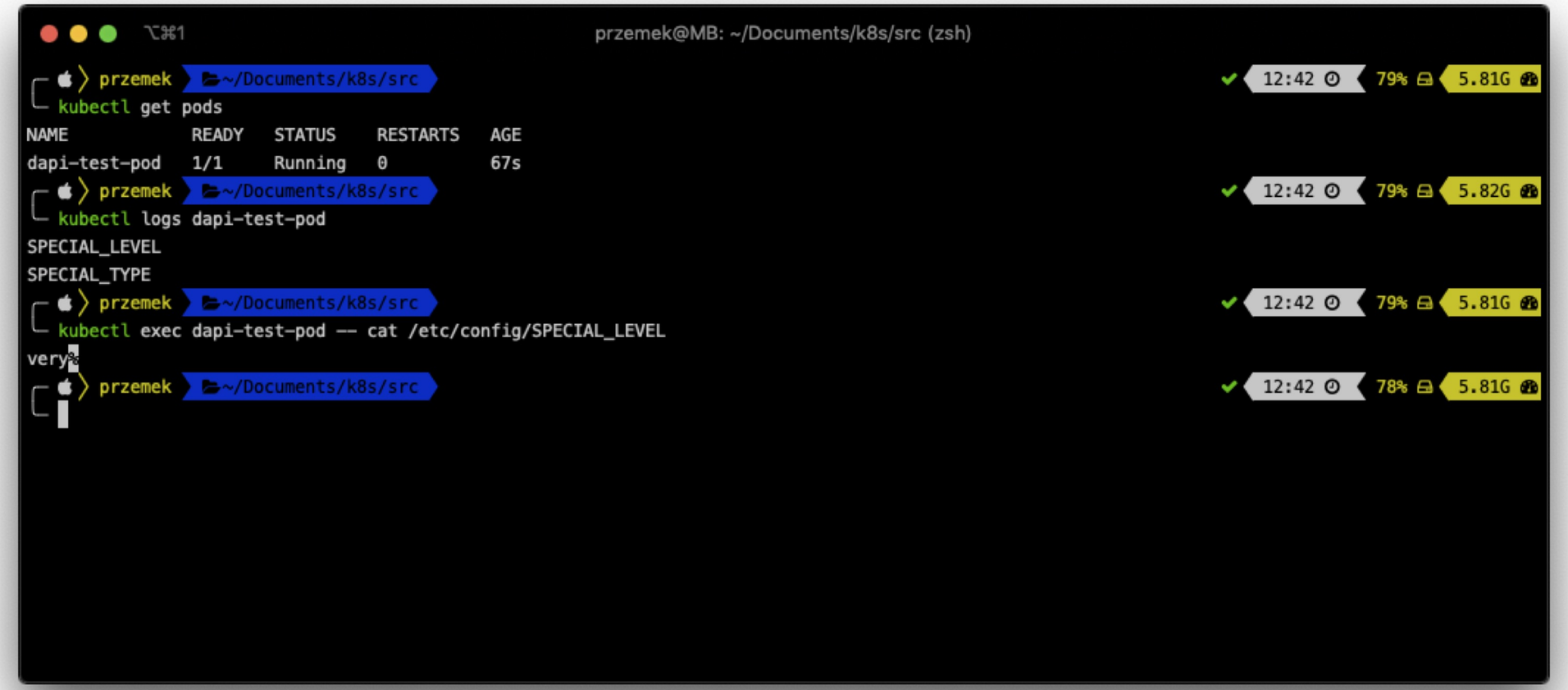
Chmurowisko

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  SPECIAL_LEVEL: very
  SPECIAL_TYPE: charm
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "ls /etc/config/ && sleep 3600" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        # Provide the name of the ConfigMap
        # containing the files you want
        # to add to the container
        name: special-config
  restartPolicy: Never
```

ConfigMaps - Volumes

Chmurowisko



Secrets

Chmurowisko

- A way of securely storing data and providing it to containers
- A Kubernetes object that stores sensitive data, such as password, keys, or tokens
- Secrets enable container images to be created without bundling sensitive data

By default, Kubernetes secrets are stored in plain text in the etcd storage for the cluster. Depending on your requirements, this may not be sufficient security for you. In particular, anyone who has cluster administration rights in your cluster will be able to read all of the secrets in the cluster.

Secrets

Chmurowisko

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret
stringData:
  myKey: mySecretPassword
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-secret-pod
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', "sleep 3600"]
    env:
    - name: MY_PASSWORD
      valueFrom:
        secretKeyRef:
          name: secret
          key: myKey
```

Secrets

Chmurowisko

Secrets – Private Docker Registries

Chmurowisko

```
$ kubectl create secret docker-registry my-image-pull-secret \
 --docker-username=<username> \
 --docker-password=<password> \
  --docker-email=<email-address>
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-image
spec:
  containers:
    - name: kuard-tls
      image: image_uri
      imagePullPolicy: Always
  imagePullSecrets:
  - name:  my-image-pull-secret
```

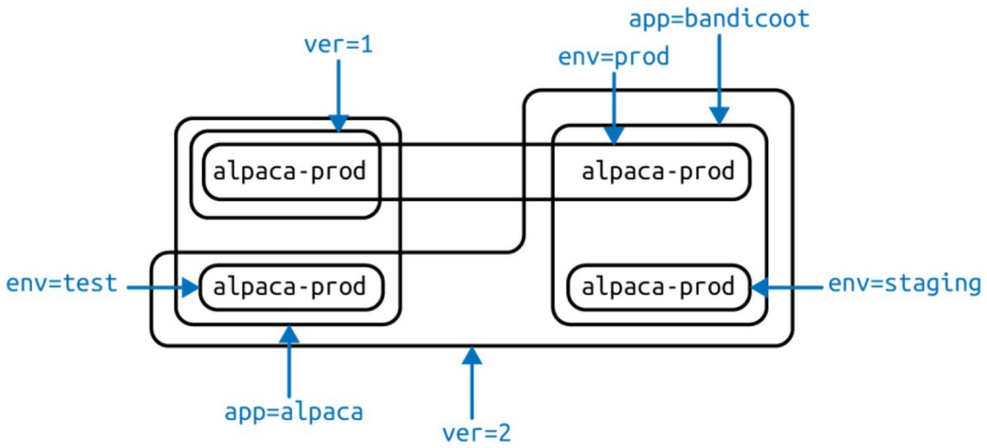
Labels

Chmurowisko

Labels are key/value pairs that can be attached to Kubernetes objects such as Pods and ReplicaSets. They can be arbitrary, and are useful for attaching identifying information to Kubernetes objects. Labels provide the foundation for grouping objects.

```
"metadata": {
  "labels": {
    "key1" : "value1",
    "key2" : "value2"
  }
}
```

```
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  labels:
    environment: production
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```



Labels do not provide uniqueness. In general, we expect many objects to carry the same label(s)

Label Selectors

Chmurowisko

- Via a *label selector*, the client/user can identify a set of objects. The label selector is the core grouping primitive in Kubernetes.
- Labels play a critical role in linking various related Kubernetes objects. In many cases objects need to relate to one another, and these relationships are defined by labels and label selectors.
- For example, ReplicaSets, which create and maintain multiple replicas of a Pod, find the Pods that they are managing via a selector. Likewise, a service load balancer finds the Pods it should bring traffic to via a selector query

Annotations

Chmurowisko

Annotations, provide a storage mechanism that resembles labels: annotations are key/value pairs designed to hold nonidentifying information that can be leveraged by tools and libraries.

```
"metadata": {
  "annotations": {
    "key1" : "value1",
    "key2" : "value2"
  }
}
```

```
apiVersion: v1
kind: Pod
metadata:
  name: annotations-demo
  annotations:
    imageregistry: "https://hub.docker.com/"
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

Annotations

Chmurowisko

Notes:

- Keep track of a “reason” for the latest update to an object.
- Communicate a specialized scheduling policy to a specialized scheduler.
- Extend data about the last tool to update the resource and how it was updated
- Build, release, or image information that isn’t appropriate for labels (may include a Git hash, timestamp, PR number, etc.).
- Enable the Deployment object to keep track of ReplicaSets that it is managing for rollouts.

Node Selector

Chmurowisko

You can constrain a Pod to only be able to run on particular Node(s), or to prefer to run on particular nodes.

Generally such constraints are unnecessary, as the scheduler will automatically do a reasonable placement (e.g. spread your pods across nodes, not place the pod on a node with insufficient free resources, etc.) but there are some circumstances where you may want more control on a node where a pod lands, e.g. to ensure that a pod ends up on a machine with an SSD attached to it, or to co-locate pods from two different services that communicate a lot into the same availability zone.

There are several ways to do this, and the recommended approaches all use label selectors to make the selection.

Node Selector

Chmurowisko

1. Attach label to the node

```
$ kubectl label nodes <node-name> <label-key>=<label-value>
```

```
$ kubectl label nodes my-kubernetes-node disktype=ssd
```

2. Add a nodeSelector field to your pod configuration

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

Resources

Chmurowisko

Every container can specify request capacity and capacity limits. Resources are requested per container, not per Pod

- Resource *requests* specify the minimum amount of a resource required to run the application.

Kubernetes guarantees that these resources are available to the Pod. The most commonly requested resources are CPU and memory, but Kubernetes has support for other resource types as well, such as GPUs and more.

- Resource *limits* specify the maximum amount of a resource that an application can consume.

When you establish limits on a container, the kernel is configured to ensure that con- sumption cannot exceed these limits. A container with a memory limit of 256 MB will not be allowed additional memory (e.g., malloc will fail) if its memory usage exceeds 256 MB.

Resources

Chmurowisko

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

Resources - CPU

Chmurowisko

One cpu, in Kubernetes, is equivalent to:

- 1 AWS vCPU
- 1 GCP Core
- 1 Azure vCore
- 1 IBM vCPU
- 1 *Hyperthread* on a bare-metal Intel processor with Hyperthreading

Core units: 100m (100 mili cpu) = 0,1 (0,1 cpu)

CPU is always requested as an absolute quantity, never as a relative quantity; 0.1 is the same amount of CPU on a single-core, dual-core, or 48-core machine.

Resources - Memory

Chmurowisko

Memory limits and requests:

- They are measured in bytes.
- You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.

Resource Quotas - Scheduler

Chmurowisko

- When you create a Pod, the Kubernetes scheduler selects a node for the Pod to run on.
- Each node has a maximum capacity for each of the resource types: the amount of CPU and memory it can provide for Pods.
- The scheduler ensures that, for each resource type, the sum of the resource requests of the scheduled Containers is less than the capacity of the node.

Resource Quotas - kubelet

Chmurowisko

- When the kubelet starts a Container of a Pod, it passes the CPU and memory limits to the container runtime.
- If a Container exceeds its memory limit, it might be terminated. If it is restartable, the kubelet will restart it, as with any other type of runtime failure.
- If a Container exceeds its memory request, it is likely that its Pod will be exited whenever the node runs out of memory.

Pod Resource Management

Chmurowisko

Notes:

- The pod is guarnted to have at least the requested resources, this is not a maximum. If there are more resources on node, the Pod will get as much as available, unless other resources are not present.
- Memory requests are handled similarly to CPU, but there is an important difference. If a container is over its memory request, the OS can’t just remove memory from the process, because it’s been allocated. Consequently, when the system runs out of memory, the kubelet terminates containers whose memory usage is greater than their requested memory. These containers are automatically restarted, but with less available memory on the machine for the container to consume.
- We can limit other resources like GPU, disks.
- With resource limit the container will never get more than the limit, if the proces inside the container will try to get more, the allocation memory funciton will fail.

Scheduling Pods

Chmurowisko

1. Does the node has adequate hardware resources?
2. Is the node running out of resources?
3. Does the pod request a specific node?
4. Does the node has a maching label?
5. If the pod requests a port, is it available?
6. If the pod requests volume, can it be mounted?
7. Does the pod tolerate the taints of the node?
8. Does the pod specify the pod or node affinity?

Pod Lifecycle

Chmurowisko

Phase	Description
Pending	The Pod has been accepted by the Kubernetes system, but one or more of the Container images has not been created. This includes time before being scheduled as well as time spent downloading images over the network, which could take a while.
Running	The Pod has been bound to a node, and all of the Containers have been created. At least one Container is still running, or is in the process of starting or restarting.
Succeeded	All Containers in the Pod have terminated in success, and will not be restarted.
Failed	All Containers in the Pod have terminated, and at least one Container has terminated in failure. That is, the Container either exited with non-zero status or was terminated by the system.
Unknown	For some reason the state of the Pod could not be obtained, typically due to an error in communicating with the host of the Pod.
Completed	The pod has run to completion as there's nothing to keep it running eg. Completed Jobs.
CrashLoopBackOff	This means that one of the containers in the pod has exited unexpectedly, and perhaps with a non-zero error code even after restarting due to restart policy.

Pod – Container Probes

Chmurowisko

A Probe is a diagnostic performed periodically by the kubelet on a Container.

To perform a diagnostic, the kubelet calls a Handler implemented by the Container.

There are three types of handlers:

- ExecAction: Executes a specified command inside the Container. The diagnostic is considered successful if the command exits with a status code of 0.
- TCPSocketAction: Performs a TCP check against the Container's IP address on a specified port. The diagnostic is considered successful if the port is open.
- HTTPGetAction: Performs an HTTP Get request against the Container's IP address on a specified port and path. The diagnostic is considered successful if the response has a status code greater than or equal to 200 and less than 400.

Pod – Container Probes

Chmurowisko

Each probe has one of three results:

- Success: The Container passed the diagnostic.
- Failure: The Container failed the diagnostic.
- Unknown: The diagnostic failed, so no action should be taken.

The kubelet can optionally perform and react to two kinds of probes on running Containers:

- *livenessProbe* - Indicates whether the Container is running. If the liveness probe fails, the kubelet kills the Container, and the Container is subjected to its restart policy. If a Container does not provide a liveness probe, the default state is Success.
- *readinessProbe* - Indicates whether the Container is ready to service requests. If the readiness probe fails, the endpoints controller removes the Pod's IP address from the endpoints of all Services that match the Pod. The default state of readiness before the initial delay is Failure. If a Container does not provide a readiness probe, the default state is Success.

Pod – Restart Policies

Chmurowisko

- A PodSpec has a restartPolicy field with possible values:
  - Always,
  - OnFailure,
  - Never.
- The default value is Always.
- restartPolicy only refers to restarts of the Containers by the kubelet on the same node.
- Exited Containers that are restarted by the kubelet are restarted with an exponential back-off delay (10s, 20s, 40s …) capped at five minutes, and is reset after ten minutes of successful execution.

Liveness Probe

Chmurowisko

Liveness probes are defined per container, which means each container inside a Pod is health-checked separately.

Notes:

Liveness probe is app specific – then it is defined at the container level.

If the Liveness probe fail, cluster will restart the pod.

If Liveness probe is not defined, cluster will only verify if the main proces is up and runing, even if the proces has deadlock it will not be restarted.

Liveness Probe

Chmurowisko

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

Readiness Probe

Chmurowisko

Readiness describes when a container is ready to serve user requests. Containers that fail readiness checks are removed from service load balancers.

Readiness probes are configured similarly to liveness probes.

```
readinessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

Configuring Probes

Chmurowisko

- *initialDelaySeconds* - Number of seconds after the container has started before liveness or readiness probes are initiated.
- *periodSeconds* - How often (in seconds) to perform the probe. Default to 10 seconds. Minimum value is 1.
- *timeoutSeconds* - Number of seconds after which the probe times out. Defaults to 1 second. Minimum value is 1.
- *successThreshold* - Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1. Must be 1 for liveness. Minimum value is 1.
- *failureThreshold* - When a Pod starts and the probe fails, Kubernetes will try failureThreshold times before giving up. Giving up in case of liveness probe means restarting the container. In case of readiness probe the Pod will be marked Unready. Defaults to 3. Minimum value is 1.

Chmurowisko

www.chmurowisko.pl

Ingress & Egress in K8s

Module Overview

Chmurowisko

- Overview
- What is ingress?
- Ingress Controller
- Real implementations
- Summary

Overview and what is ingress?

Chmurowisko

- Service is treated as the way of exposing set of pods (using label selectors) through virtually one IP which is routable inside the cluster
- An API object that manages external access to the services in a cluster, typically HTTP.

- **Ingress** solves problems like:
  - Externally-reachable URLs
  - Load Balance traffic
  - SSL / TLS termination
  - name based hosting

- **Ingress Controller** can implement ingress object by providing some real, technical implementation

Sample Ingress object

Chmurowisko

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        backend:
          serviceName: test
          servicePort: 80
```

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/force-ssl-redirect: "false"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  tls:
  - hosts:
    - chmurowisko.pl
    secretName: tls-secret
  rules:
#  - host: chmurowisko.pl
   - http:
      paths:
        - path: /srv1
          backend:
            serviceName: service1-service
            servicePort: 5678
        - path: /srv2
          backend:
            serviceName: service2-service
            servicePort: 5678
```

Ingress Controller

Chmurowisko

- There are plenty of ways to implement ingress through various solutions
- There is no ingress delivered by kube-controller-manager when the cluster is created
- As a project K8s supports **nginx (**https://kubernetes.github.io/ingress-nginx/deploy/) and **GCE** controller (by official project)
- There are plenty of implementations:
  - Kong
  - Traefik
  - Istio
  - HAProxy
  - Citrix
  - Ambassador
  - and more…

Ingress Controller should implement one standard but this is not the case in all situations.

- You can deploy many ingress controllers to one cluster if you like.

Types of ingress

Chmurowisko

- Single Service Ingress – one front-end route to one backend
- Fanout – one front-end routes to two backend’s
- Name based routes (based on host headers)
- TLS
- Load Balancing

You don’t need ingress controller to expose service:

- Service.Type = LoadBalancer
- Service.Type = NodePort

Egress

Chmurowisko

- A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.
- **NetworkPolicy** resources use labels to select pods and define rules which specify what traffic is allowed to the selected pods.
- **NetworkPolicy** may include a list of whitelist egress rules.
- Each rule allows traffic which matches both the to and ports sections.
- Network policy defines **Ingress and Egress** rules.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
spec:
  podSelector: {}
  egress:
  - {}
  policyTypes:
  - Egress
```

Network Policies

Chmurowisko

- Network policies are used to control traffic flow at the IP address or port level (OSI layer 3 or 4)
- They specify how a pod is allowed to communicate with various network "entities”
- An entity can be:
  - Another pod (via selector)
  - Namespace (via selector)
  - IP blocks
- You need a network plugin to use network policies:
  - Calico
  - Azure Network policy
  - Canal
  - Cilium
  - And others…

Network Policies

Chmurowisko

- By default all pods can communicate
- When a pod has a network policy it becomes isolated. You need to allow connection to this pod in the network policy definition
- There are two policy types
    - Ingress
    - Egress

Network Policies

Chmurowisko

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

Network Policies

Chmurowisko

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress
spec:
  podSelector: {}
  ingress:
  - {}
  policyTypes:
  - Ingress
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-egress
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Network Policies - limitations

Chmurowisko

What you **cannot** do with network policies:

- Enforce common gateway for traffic
- TLS
- Node policies
- Logging
- Explicit deny policies

Ingress vs Load Balancer vs NodePort

Chmurowisko

- The output of all of those options is the same – **they help you to expose the service externally**
- **NodePort** – it will allocate the specific port on each node and the traffic which goes through that port will be forwared to the service
- **LoadBalancer** – this is usually implemented by the Load Balancer by Cloud Provider
- **Ingress** – it is totally independent from Service, you create this independently from Service, it consolidates the whole rules in one place and brings more benefits. The config is a dedicated object.

In simple cases LoadBalancer is fine but Ingress is what you really need in most cases especially if you need app routing.

Summary

Chmurowisko

- Overview
- What is ingress?
- Ingress Controller
- Real implementations
- Summary

Chmurowisko

www.chmurowisko.pl

Storage in Kubernetes

Module Overview

Chmurowisko

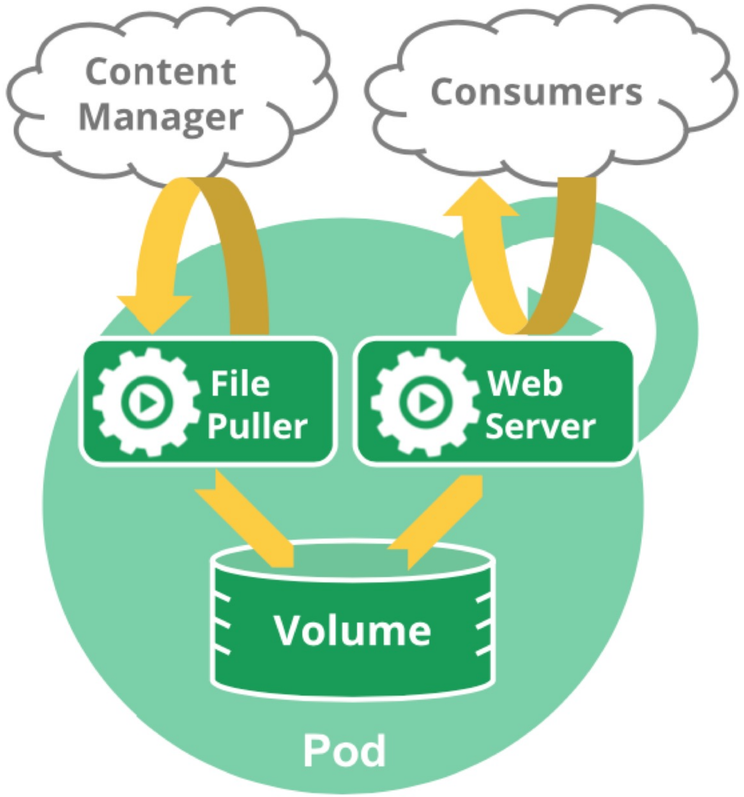
- Basic concepts
- Storage options
  - Volume
  - Storage Class
  - Persistent Volume
  - Persistent Volume Claim
- How it works and what to choose!
- Summary

Storage in Kubernetes

Chmurowisko

Let’s review some basic concepts

- Containers in pod **DO NOT** share storage
- Every container has it’s own isolated file system, every new container start’s with the new file system. File system is image based.
- Based on container lifecycle, anything that container will save in file system will be lost.
- If you want to persist not exactly the container file system, but some data you work on, you may specific K8s objects. **Some of them don't share the lifecycle of container**
- If the pod will have many containers, the **specified volume can be used by all of them,** but needs to be mounted purposefully.



Basic concepts

Chmurowisko

- Volume
- Storage Class
- Persistent Volume
- Persistent Volume Claim

Volume

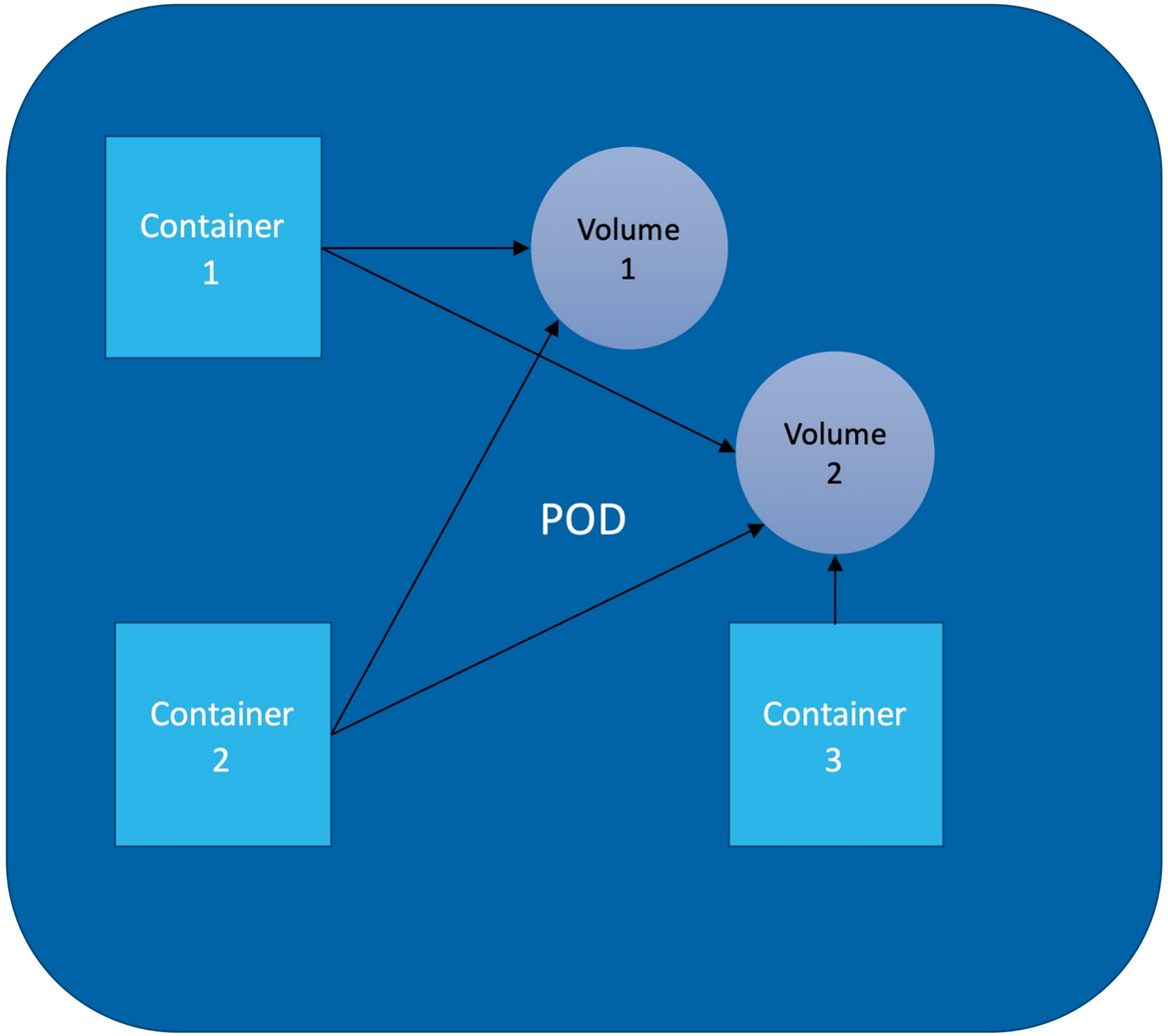
Chmurowisko

- Volume is **not created, nor deleted by it's own**
- Volume is a pod component, they are defined inside pod
- Every container must mount the volume to use it, it can be mount to any path inside
- Being in the same container is not enough

You can start with empty volume by using emptyDir as a volume type.

Basic volume diagram

Chmurowisko



Volume Types

Chmurowisko

- **emptyDir** - an empty directory
- **hostPath** - used for mounting directories from the worker node's filesystem into the pod (hope you remember our Docker story around storage still)
- **gitRepo** - volume initialized by checking out the contents of a Git repository.
- **nfs -** an NFS share
- **Cloud Volume Types**:
  - gcePersistentDisk
  - awsElastic-BlockStore
  - azureDisk
- More of those:
  - cinder, cephfs, iscsi ....
  - other network drivers
- **persistentVolumeClaim**—A way to use a pre- or dynamically provisioned persistent storage

- GCEPersistentDisk
- AWSElasticBlockStore
- AzureFile
- AzureDisk
- CSI
- FC (Fibre Channel)
- FlexVolume
- Flocker
- NFS
- iSCSI
- RBD (Ceph Block Device)
- CephFS
- Cinder (OpenStack block storage)
- Glusterfs
- VsphereVolume
- Quobyte Volumes
- HostPath (Single node testing only – local :
- Portworx Volumes
- ScaleIO Volumes
- StorageOS

Volume Types

Chmurowisko

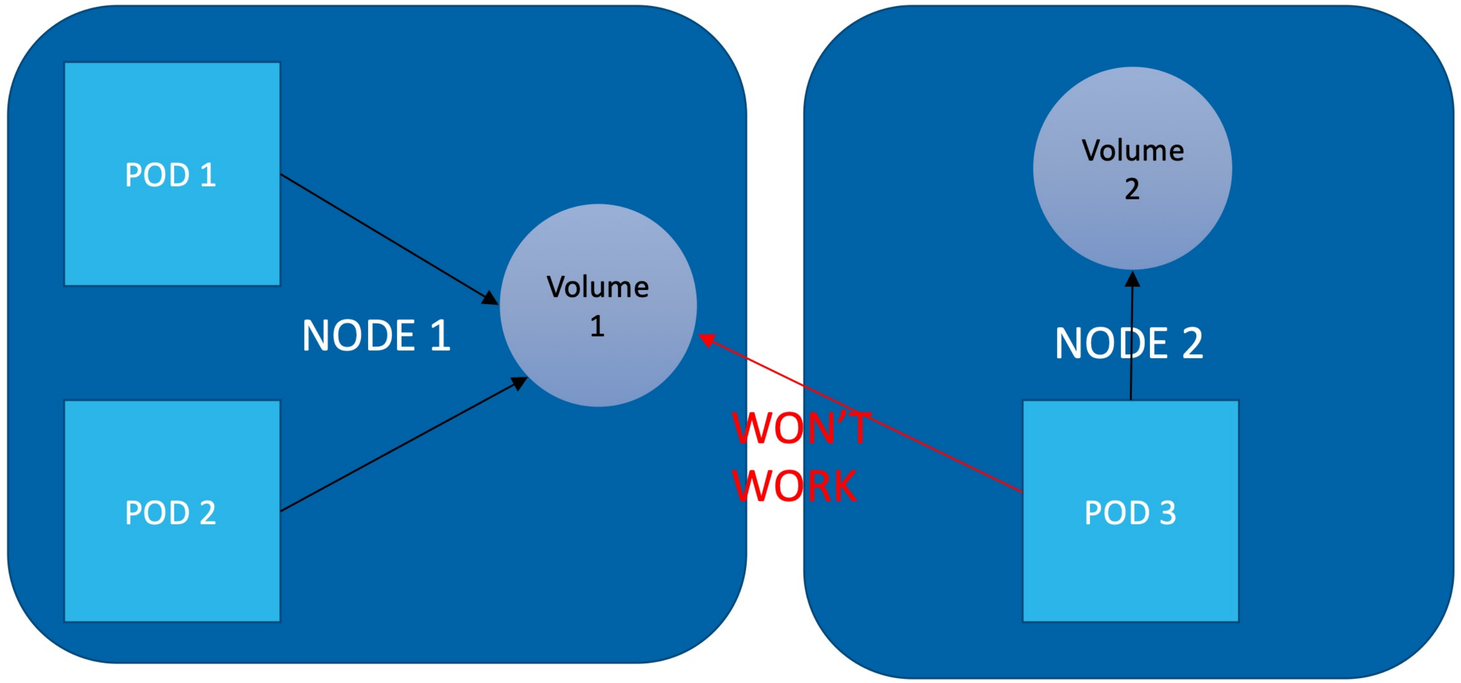
We also have some special types:

- **configMap**, secret – way to map specific K8s objects to pod as the volume
- **persistentVolumeClaim** – way to use precreated or dynamically create persistent storage (we will come back to that)

HostPath Volume

Chmurowisko

- **hostPath** volumes - first type of persistent storage, it will remain even if the pod will be deleted becuase it will remian on host
- **gitRepo** and **emptyDir** volumes' will be deleted, one the pod will be gone
- If pod uses hostPath volume pointing to the same path on the host, the new pod will see content of the previous one (if only on the same node)



hostPath is not perfect ☺

if the pod's get rescheduled, the pod will see nothing, it will not be able to mount the volume. **Based of K8s principals this is not you want to have!**

Let’s see some examples in action

Chmurowisko

volumes:

- name: html

emptyDir:

medium: Memory

```
apiVersion: v1
kind: Pod
metadata:
  name: gitrepo-volume-pod
spec:
  containers:
  - image: nginx:alpine
    name: web-server
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
      readOnly: true
    ports:
    - containerPort: 80
      protocol: TCP
  volumes:
  - name: html
    gitRepo:
      repository: https://github.com/luksa/kubia-website-example.git
      revision: master
      directory: .
```

You’re creating a gitRepo volume.

The volume will clone this Git repository.

The master branch will be checked out.

You want the repo to be cloned into the root dir of the volume.

Pod with volume

Chmurowisko

```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  volumes:
  - name: mongodb-data
    gcePersistentDisk:
      pdName: mongodb
      fsType: ext4
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
    ports:
    - containerPort: 27017
      protocol: TCP
```

The name of the volume (also referenced when mounting the volume)

The type of the volume is a GCE Persistent Disk.

The name of the persistent disk must match the actual PD you created earlier.

The filesystem type is EXT4 (a type of Linux filesystem).

The path where MongoDB stores its data

Persistent Volume & Persistent Volume Claim

Chmurowisko

- None of the concepts shown is not perfect, if you **want to persist data and access them even if you get rescheduled** to other node
- More over **if the data needs to be accessed from many pods on various nodes**, you need the other concept

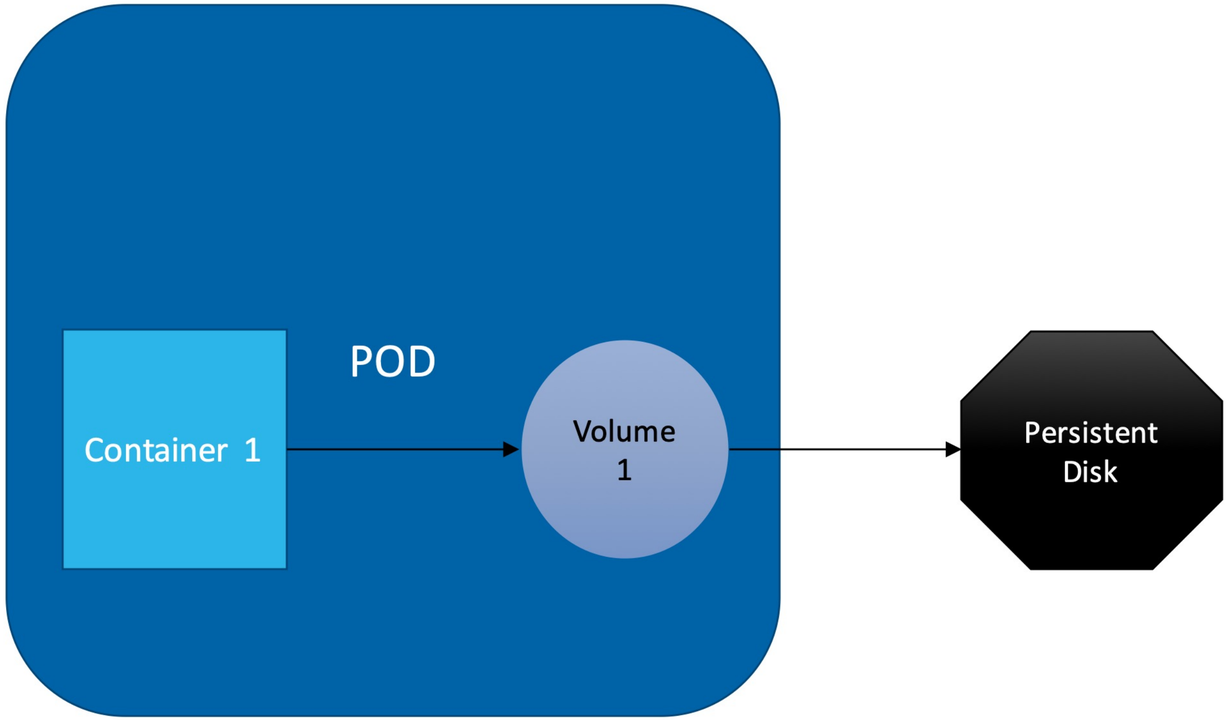
```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
  creationTimestamp: null
  labels:
    kubernetes.io/cluster-service: "true"
  name: managed-premium-szopex
  selfLink: /apis/storage.k8s.io/v1/stora
parameters:
  cachingmode: ReadOnly
  kind: Managed
  storageaccounttype: Premium_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-butydlamalucha2
  labels:
    type: nfs
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.253.7
    path: /drbd/data
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: butydlamalucha2-pv-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
```

Chmurowisko

Pod with volume on Persistent Disk

Chmurowisko



StorageClass definition

Chmurowisko

```
└─ kubectl get storageclass
NAME                    PROVISIONER                AGE
azurefile               kubernetes.io/azure-file   41h
azurefile-premium       kubernetes.io/azure-file   39h
default (default)       kubernetes.io/azure-disk   13h
managed-premium         kubernetes.io/azure-disk   13h
managed-premium-szopex  kubernetes.io/azure-disk   47h
```

```
└─ kubectl describe storageclass/managed-premium
Name:            managed-premium
IsDefaultClass:  No
Annotations:     kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"storage.k8s.io/v1beta1","kind":"StorageClass","metadata":{"annota
tions":{},"labels":{"kubernetes.io/cluster-service":"true"},"name":"managed-premium"},"parameters":{"cachingmode":"ReadOnly","kind":"Managed","sto
rageaccounttype":"Premium_LRS"},"provisioner":"kubernetes.io/azure-disk"}

Provisioner:           kubernetes.io/azure-disk
Parameters:            cachingmode=ReadOnly,kind=Managed,storageaccounttype=Premium_LRS
AllowVolumeExpansion:  <unset>
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>
```

Persistent Volume & Persistent Volume Claim

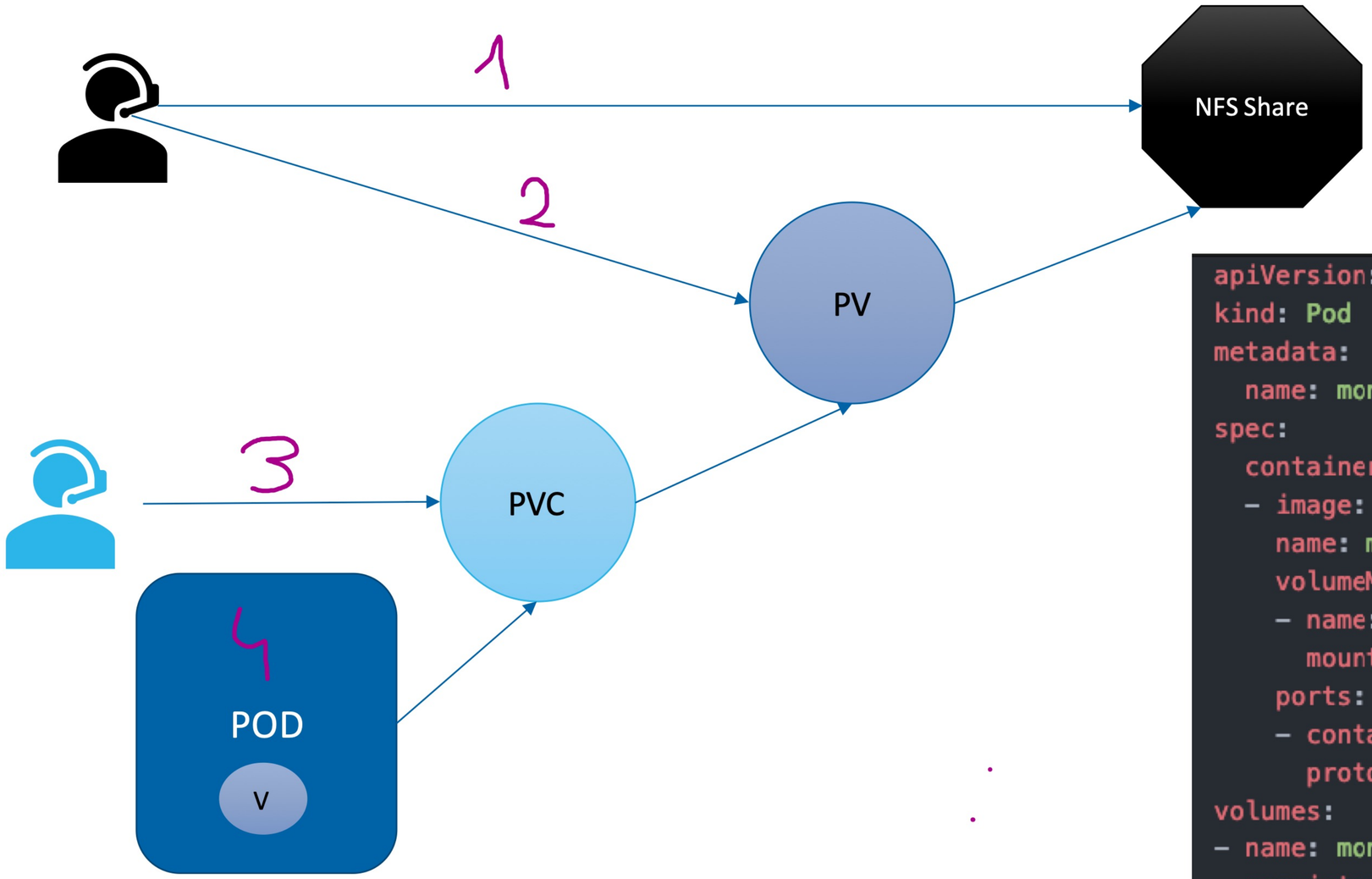
Chmurowisko

- **Persistent Volume (PV)**
  - Storage provisioned by admin of cluster using specific Storage Class
  - Resource of the cluster like any other (pod, node, service)
  - PV's lifecycle is not Pod related, if you delete the Pod the PV will remain
  - Dedicated driver handels the way we work with particular storage

- **Persistent Volume Claim (PVC)**
  - Storage requested by the user while creating the Pod
  - You (as requestor) defines what kind of storage you want to use – size / access mode (read/write once, read many times)
  - PVC's hides the implementation from You
  - Admin needs to define various Storage Class to offer different options

Let me show you the logic!

Chmurowisko




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
    ports:
    - containerPort: 27017
      protocol: TCP
volumes:
- name: mongodbdata
  persistentVolumeClaim:
    claimName: mongodbdata-pvc
```

Persistent Volume – Two types

Chmurowisko

- **Static**
  - Cluster admin creates them as resources
  - They are available for usage
  - They are defined upfront

- **Dynamic**
  - If the cluster does not have PV to match your PVC then you it can try to creat this for you
  - Then, the PVC must define the StorageClass which will be used
  - The admin can define the default storage class and you will see that Cloud Providers have at least one

Types of mounting

Chmurowisko

- ReadWriteOnce (RWO)
- ReadOnlyMany (ROX)
- ReadWriteMany (RWX)

Volume can be mounted using one access mode at one time. Even if the type supports many mounts they can be mounted using one type at once.

Phases of volume mount

Chmurowisko

- Available
- Bound
- Released
- Failed

```
└─ kubectl get pvc --all-namespaces
NAMESPACE   NAME                          STATUS   VOLUME                CAPACITY   ACCESS MODES   STORAGECLASS   AGE
tests       b[redacted]2-pv-claim         Bound    nfs-[redacted]a2      5Gi        RWX                           12h
tests       c[redacted]claim              Bound    nfs-[redacted]        5Gi        RWX                           12h
tests       d[redacted]pv-claim           Bound    nfs-[redacted]        5Gi        RWX                           11h
tests       d[redacted]claim              Bound    nfs-[redacted]        5Gi        RWX                           11h
tests       k[redacted]v-claim            Bound    nfs-[redacted]        5Gi        RWX                           11h
tests       s[redacted]pv-claim           Bound    nfs-[redacted]        5Gi        RWX                           11h
tests       s[redacted]y-pv-claim         Bound    nfs-[redacted]ty      5Gi        RWX                           11h
tests       s[redacted]a-pv-claim         Bound    nfs-[redacted]za      5Gi        RWX                           11h
tests       s[redacted]-pv-claim          Bound    nfs-[redacted]a       5Gi        RWX                           11h
tests       s[redacted]pv-claim           Bound    nfs-[redacted]        5Gi        RWX                           11h
tests       w[redacted]aim                Bound    nfs-[redacted]        5Gi        RWX                           11h
tests       z[redacted]im                 Bound    nfs-[redacted]        5Gi        RWX                           11h
```

Volume Lifecycle when you are done with PVC

Chmurowisko

Reclaiming policy tells the cluster what to do when the claim has been released.

The volume can be:

- Retained
- Recycled (deprecated)
- Deleted

Be carefull!

Chmurowisko

- There are various types of storage
- They behave differently
- If you will choose specifi type it may have specific limits
- Let me give you example:
  - You work with Azure
  - You want to use SSD disk and then…
  - So you choose Azure Files but they have two tiers
  - And then you want high perf. and you are directed to NetApp

> **Note**
>
> An Azure disk can only be mounted with *Access mode* type *ReadWriteOnce*, which makes it available to only a single pod in AKS. If you need to share a persistent volume across multiple pods, use Azure Files.

There is a limit of volumes per node

Chmurowisko

- Volumes may seem like a solution in many cases, however my advice always will be, if you can use something outside the K8s to persist data like files or use DBaaS, use it.
- It will will be way easier and you will decouple compute from storage option
- If you need to stay with K8s for storage (for some reasons) bear in mind that we have some limites for numer of volumes per node (AWS – 39, GCP – 16, Azure – 16)

Summary

Chmurowisko

- Basic concepts
- Storage options
  - Volume
  - Storage Class
  - Persistent Volume
  - Persistent Volume Claim
- How it works and what to choose!
- Summary