Container, Orchestration, Service Mesh

Automation for your Container Cluster

Andy Wirtz

27. April 2019

Andy Wirtz

Andy:

- IT Consultant at ATIX AG, Germany
- automation of data centers
- deployment of cloud native services
- expertise in Docker, Kubernetes, Istio

Contact:

- phone: +49 (0)89 452 35 38-248
- mail: wirtz@atix.de
- www.xing.com/profile/Andy_Wirtz2
- www.linkedin.com/in/andy-wirtz

India

Focus questions

How is it possible that my requested service:

- gets as fast and as many new features?
- is always and instantly available?
- looks sometimes diffrent for me than for my mate?

Focus questions

How is it possible that my requested service:

- gets as fast and as many new features?
- is always and instantly available?
- looks sometimes diffrent for me than for my mate?

Focus questions

How is it possible that my requested service:

- gets as fast and as many new features?
- is always and instantly available?
- looks sometimes diffrent for me than for my mate?

Agenda

1. Microservices/Container
2. Scheduler/Orchestrator
3. Service Mesh

Agenda

1 Microservices/Container

2 Scheduler/Orchestrator

3 Service Mesh

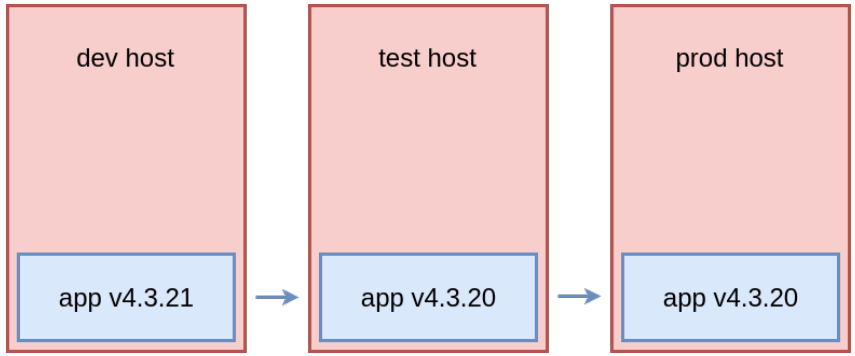
Wish list

End user:

- how is it possible that my requested service gets as fast and as many new features?

DevOps – I want:

- N deployments per day
- independence of my deployments from:
  - hardware
  - operation System
  - other applications



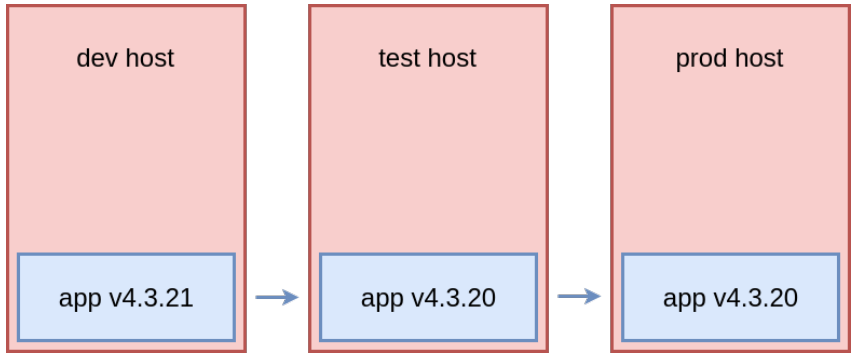
Wish list

End user:

- how is it possible that my requested service gets as fast and as many new features?

DevOps – I want:

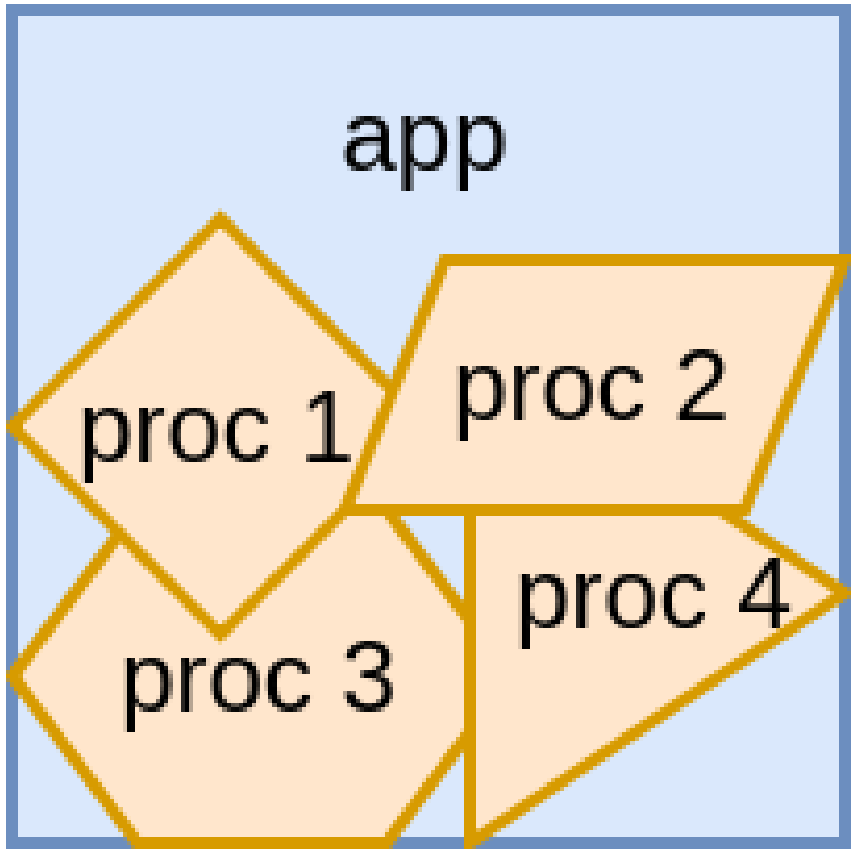
- N deployments per day
- independence of my deployments from:
  - hardware
  - operation System
  - other applications



Limit 1

Software monolith:

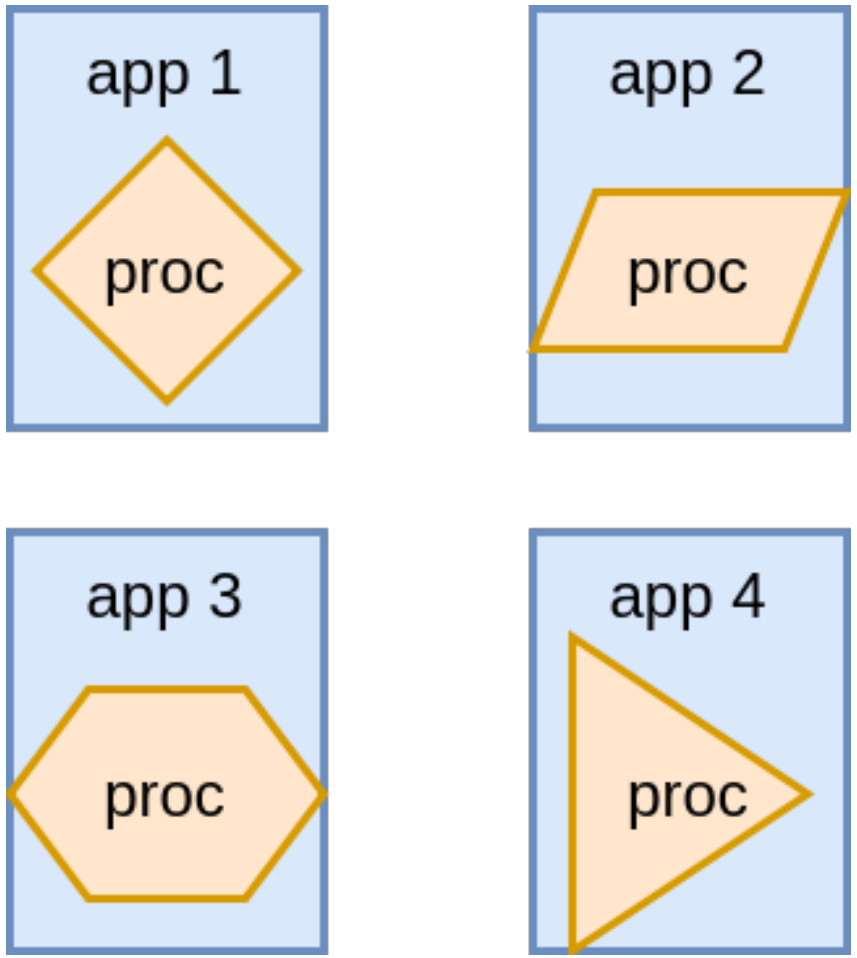
- many interlocked processes per application
- difficult to:
  - implement new features
  - get new versions running



Idea 1

Microservices:

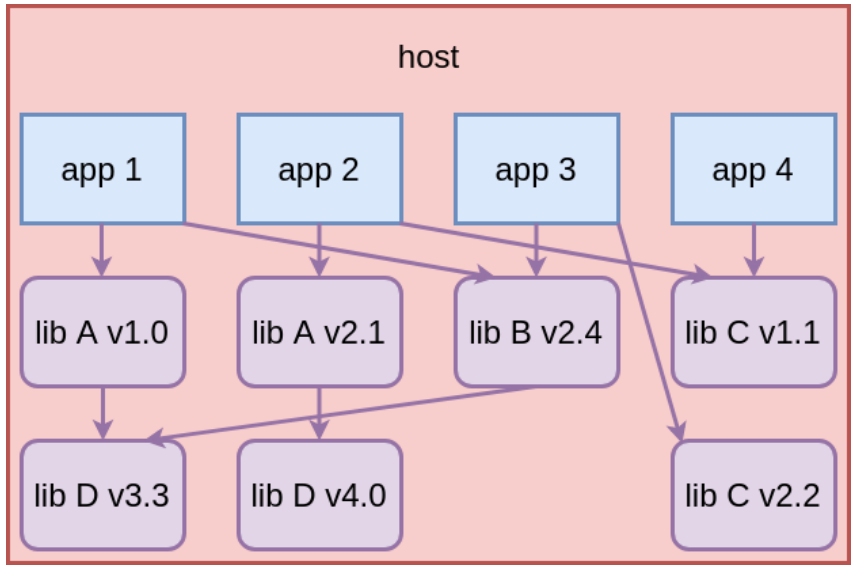
- one process per application
- communication via simple, well-defined APIs
- faster and easier feature implementation



Limit 2

Dependencies:

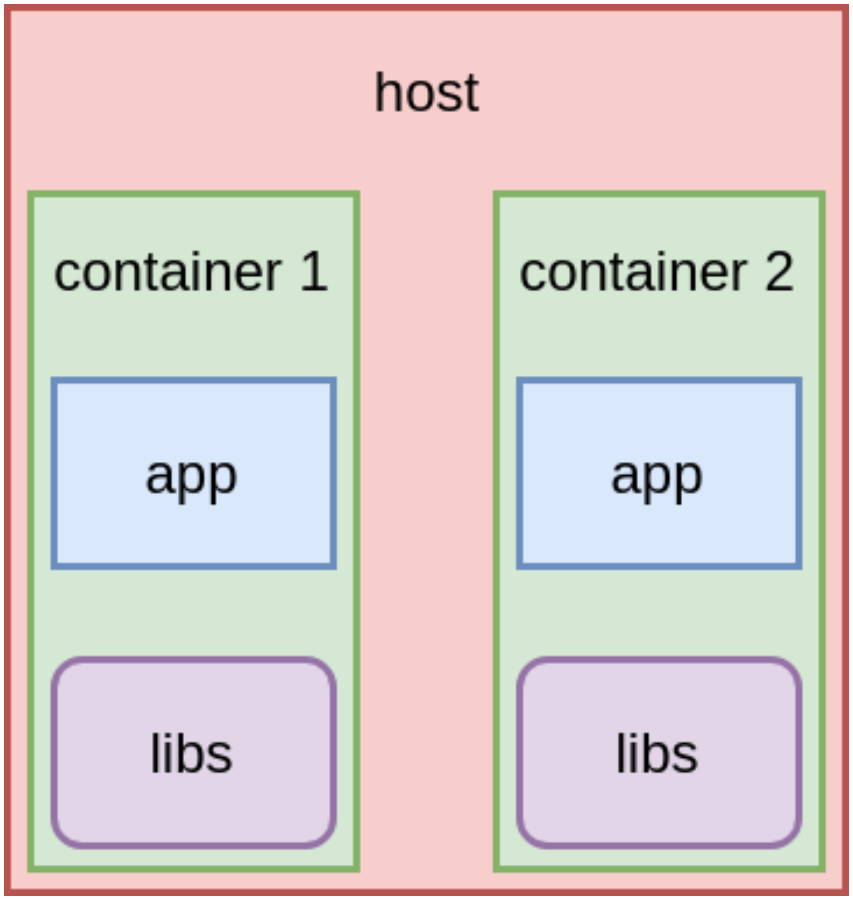
- shared use of libraries
- intransparent and error-prone
- deployment dependent on:
  - hardware
  - operation system
  - other applications



Idea 2

Container:

- application plus needed libraries
- isolation
- independence



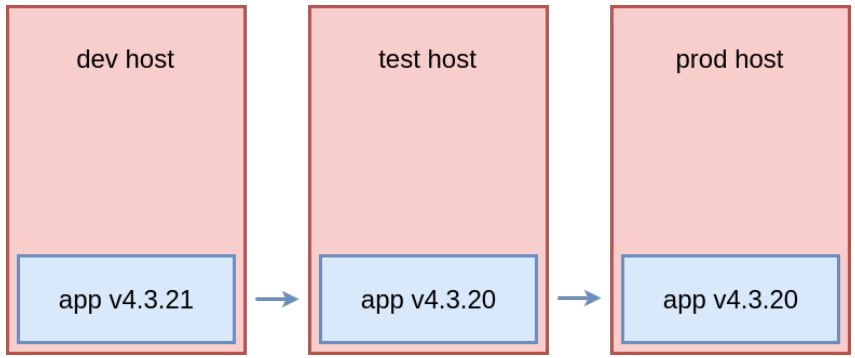
New possibilities

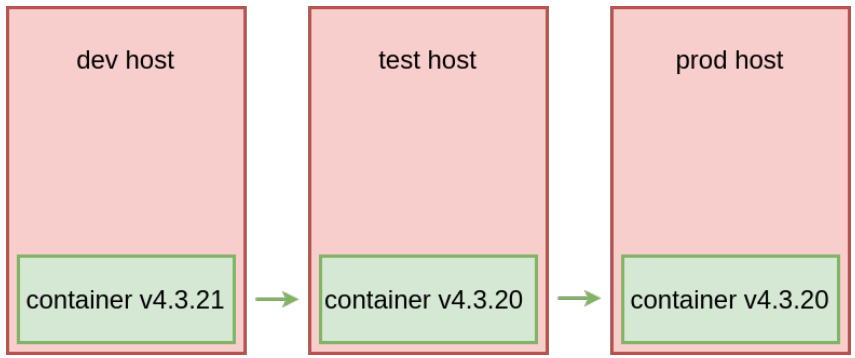
Velocity:

- for development
- for testing
- for deployment

Isolation:

- portability
- no dependency conflicts





Agenda

1 Microservices/Container

2 Scheduler/Orchestrator

3 Service Mesh

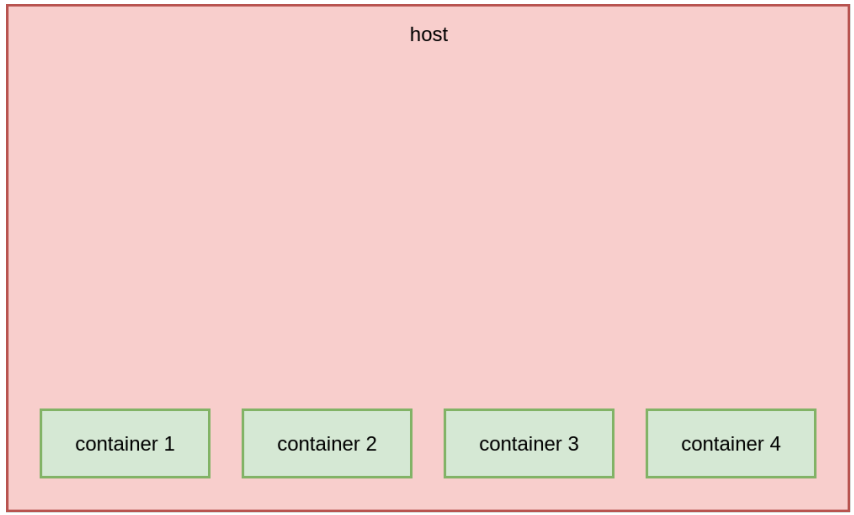
Wish list

End user:

- how is it possible that my requested service is always and instantly available?

DevOps – I want:

- to use over 80% of my hardware at any given time
- my services up and running at any given time



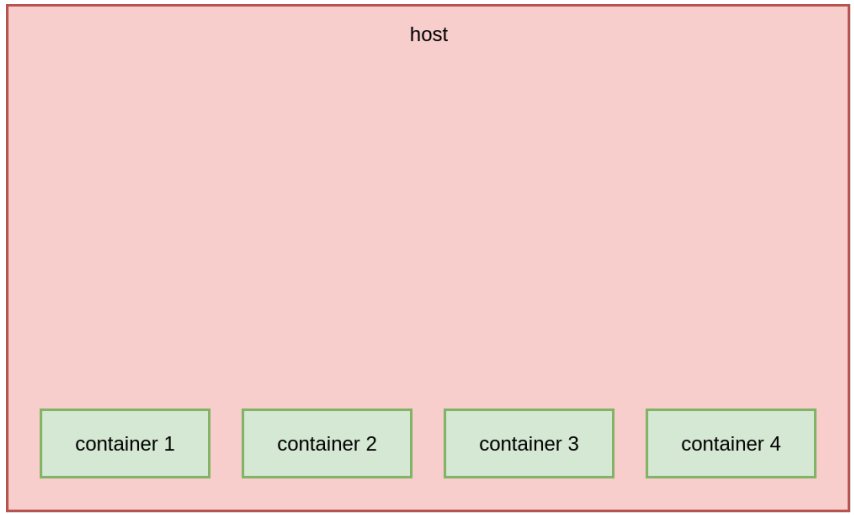
Wish list

End user:

- how is it possible that my requested service is always and instantly available?

DevOps – I want:

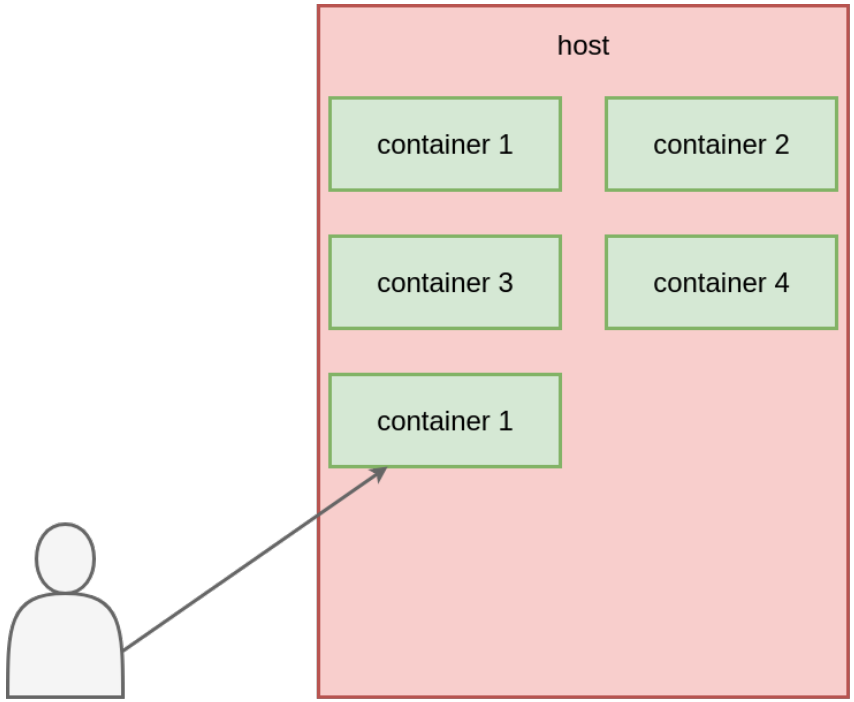
- to use over 80% of my hardware at any given time
- my services up and running at any given time



Limit 1

Static deployments:

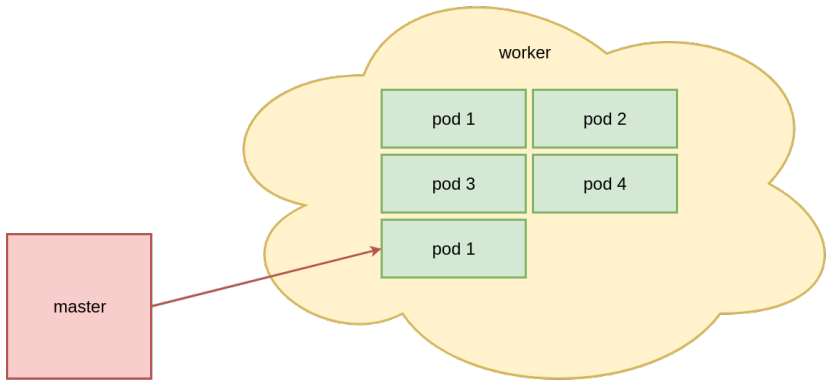
- limit resource usage of containers is predefined
- hosts with significant overhead
- deployment independent of service requests
- manual scaling as necessary



Idea 1

Abstraction of resources:

- collection of host to one cloud
- deployment of pods (containers) in this cluster



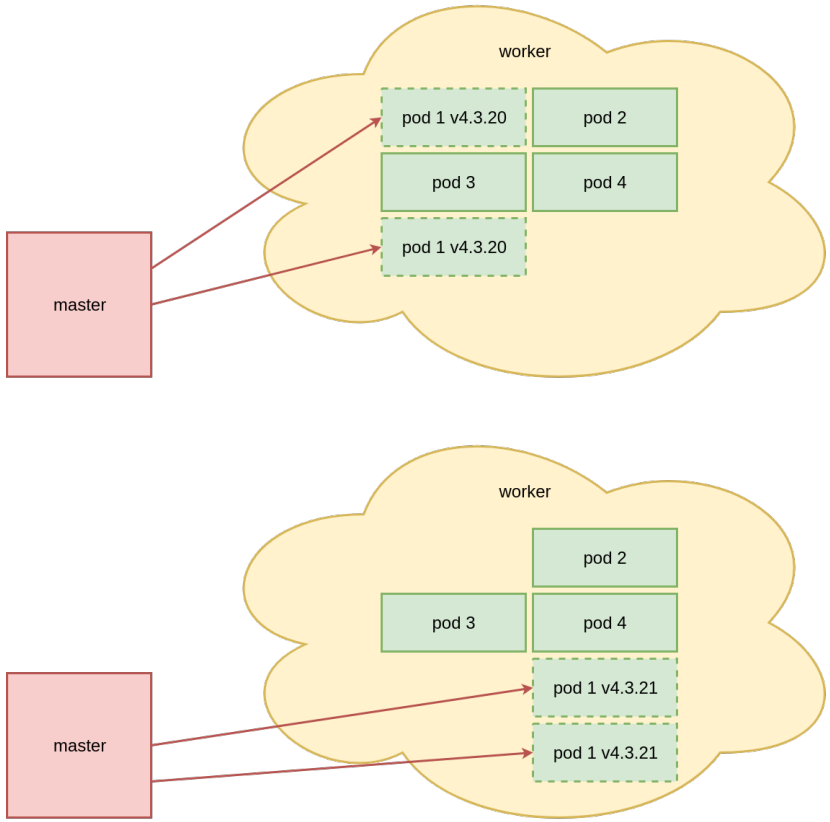
Orchestrator:

- dynamic scheduling of pods
- defining the desired state
- automatic adjustment of the current state if necessary
- automatic scaling as necessary

Limit 2

Updates with downtime:

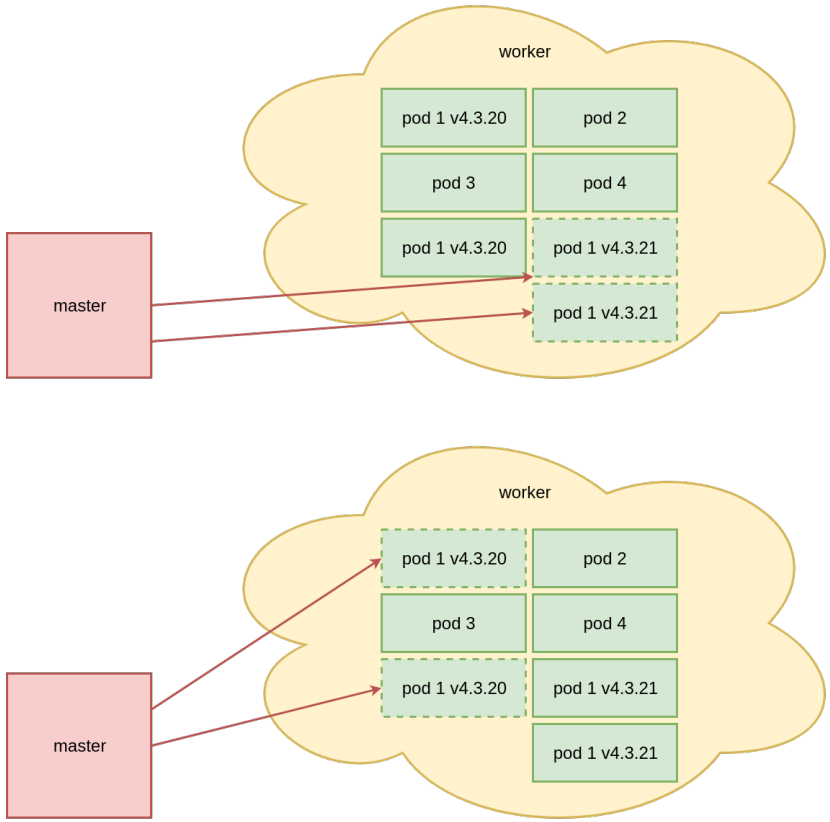
- all existing pods are killed
- new ones are created
- the service is down for a small amount of time



Idea 2

Rolling updates:

- new pods are created
- they are checked for readyness
- old wones are killed
- zero downtime



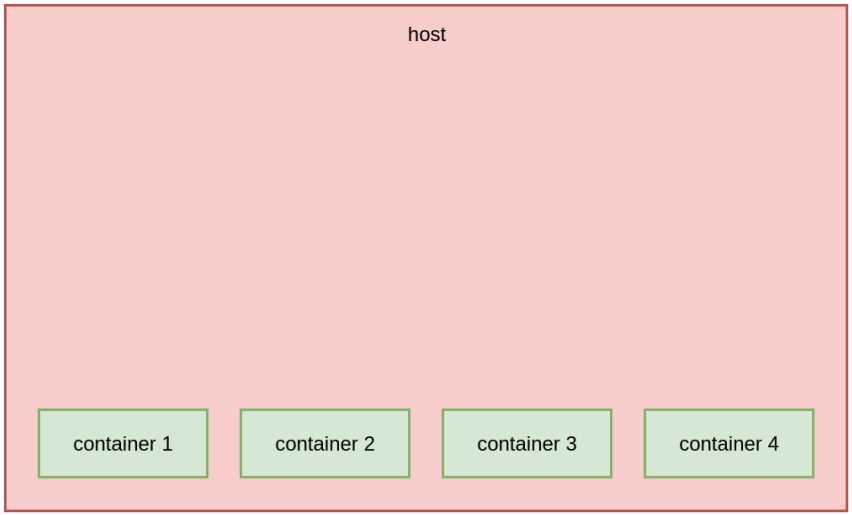
New possibilities

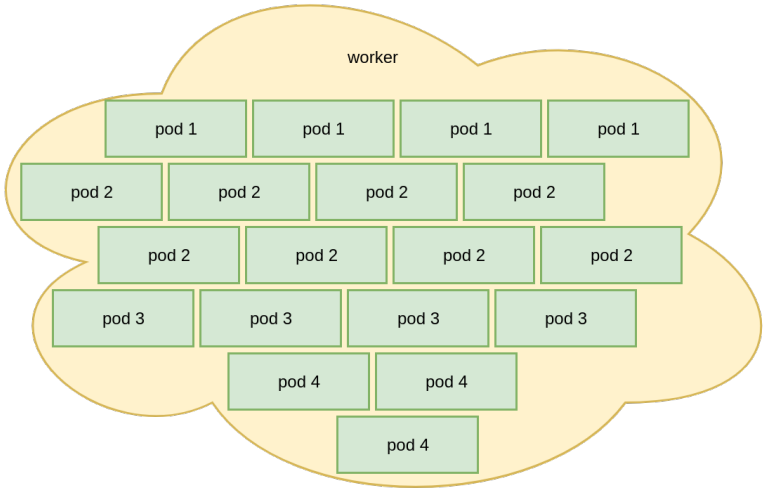
Efficiency:

- hardware abstraction
- multiple services per cloud

Automation:

- scaling dependent on the requests
- some ready to use pods in stock
- self-healing
- updates without downtime
- standardization and reproducibility





Agenda

1 Microservices/Container

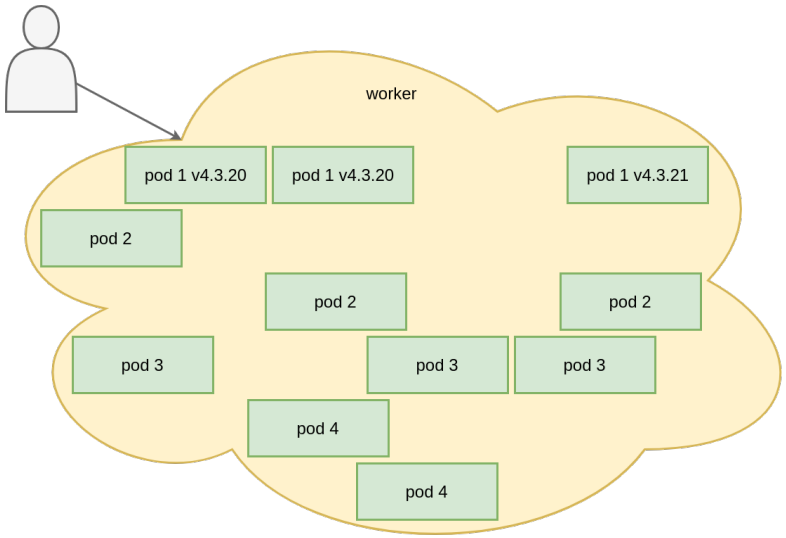
2 Scheduler/Orchestrator

3 Service Mesh

Wish list

End user:

- how is it possible that my requested service looks sometimes diffrent for me than for my mate?



DevOps – I want:

- to visualize the traffic between my services
- to deny access between some services
- to encrypt the communication between my services
- to let some end users test a new version of my service

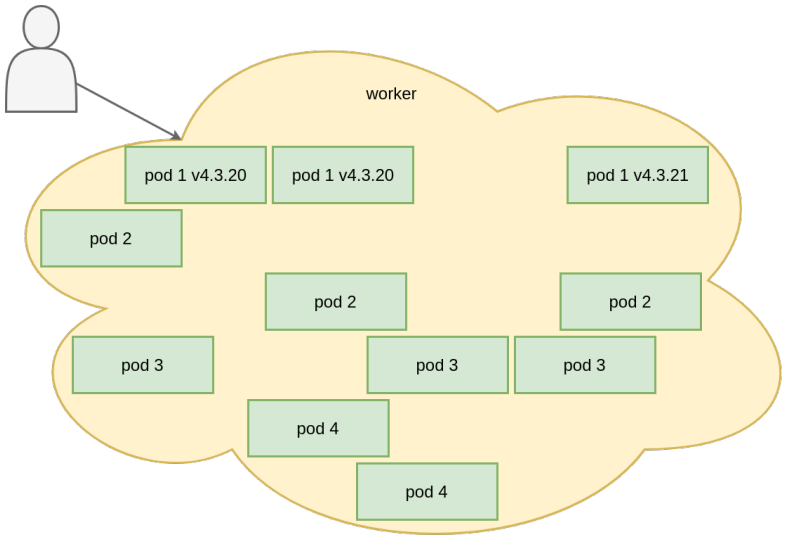
Wish list

End user:

- how is it possible that my requested service looks sometimes diffrent for me than for my mate?

DevOps – I want:

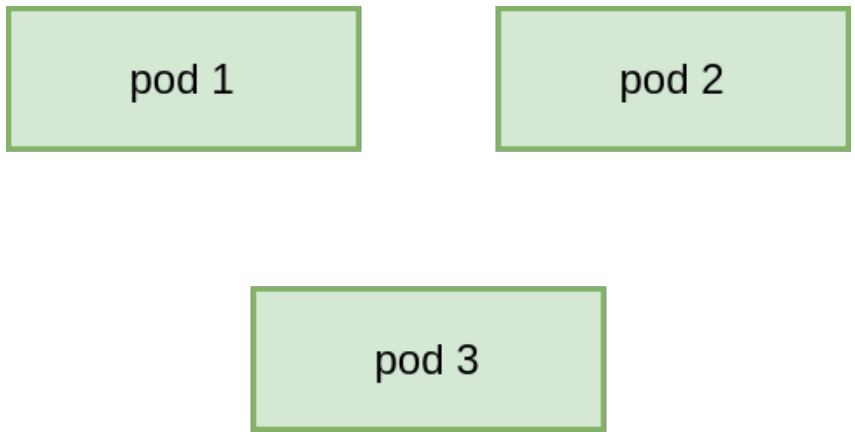
- to visualize the traffic between my services
- to deny access between some services
- to encrypt the communication between my services
- to let some end users test a new version of my service



Limit 1

Big cloud:

- possibly millions of containers in one cluster
- losing track of the connections and the communication



Idea 1: Telemetry capturing

Service Mesh:

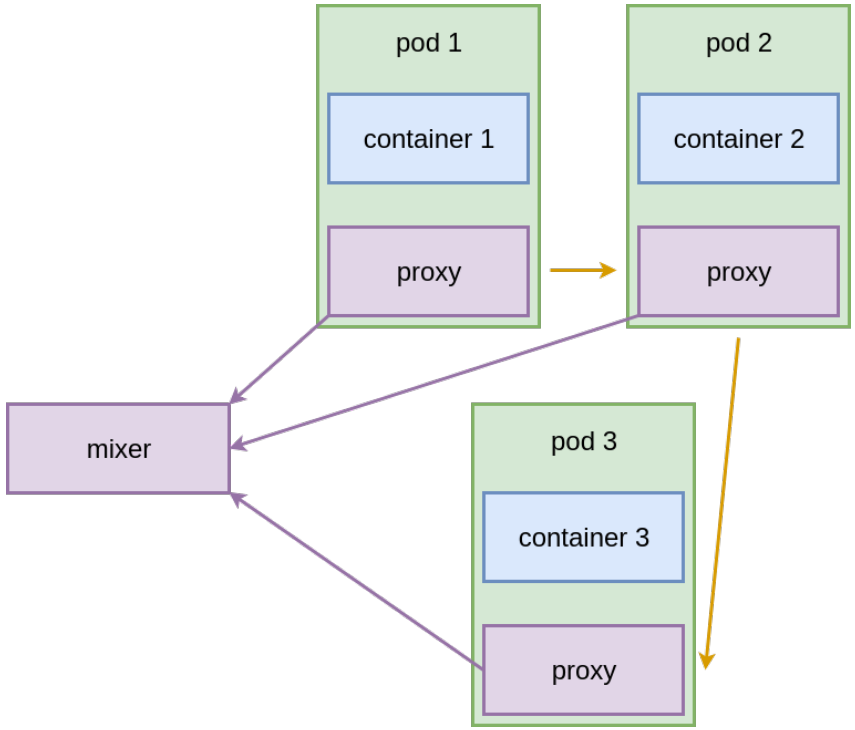
- data plane with proxies as side car
- control plane with pilot, citadel and mixer

Proxies:

- take over all network communication
- send telemetry, logs, metrics

Mixer:

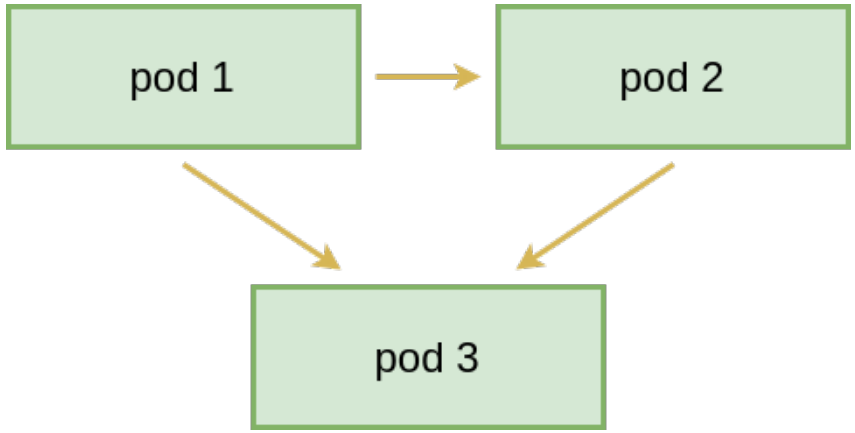
- collects telemetry, logs, metrics
- has adapters for the respective backends



Limit 2

Permissions:

- every container is allowed to reach every other container
- role based access control is possible
- no restrictions with respect to labels, attributes, ip-adresses



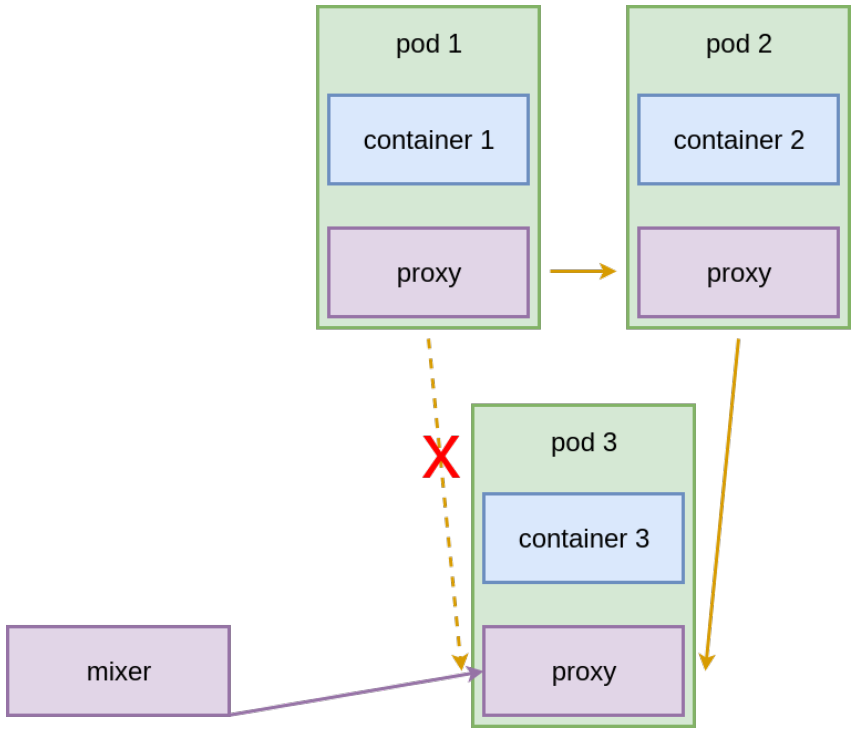
Idea 2: Policy enforcement

Proxies:

- call Mixer before each request
- perform precondition checks from cache

Mixer:

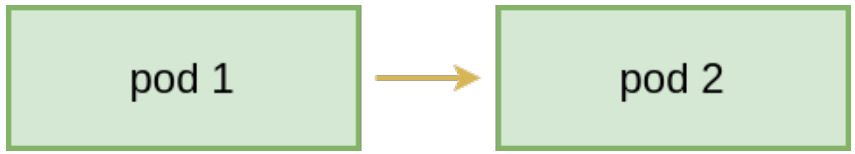
- has adapter for policy backends
- performs precondition checks



Limit 3

Network security:

- different attributes from different software defined networks
- often times no encryption of traffic



Idea 3: Security features

Proxies:

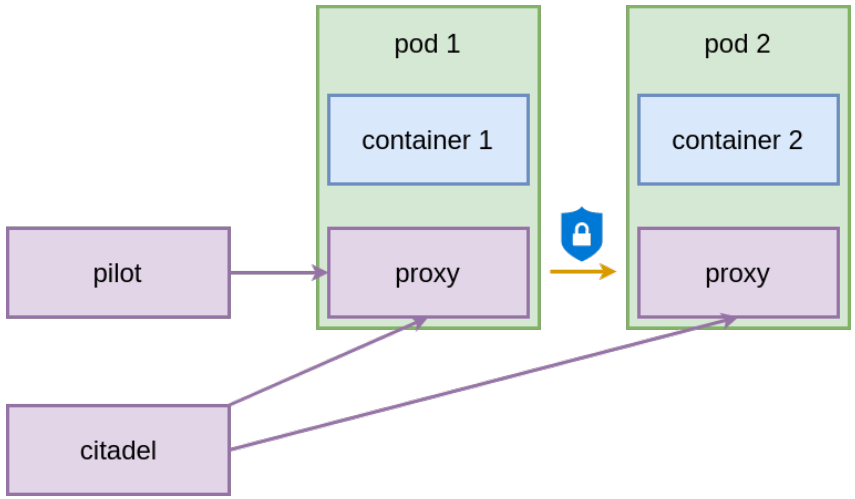
- get certificates, keys and secure naming
- perform mutual TLS communication

Citadel:

- creates certificate and key pairs
- stores them as Kubernetes secrets
- automatically rotates them

Pilot:

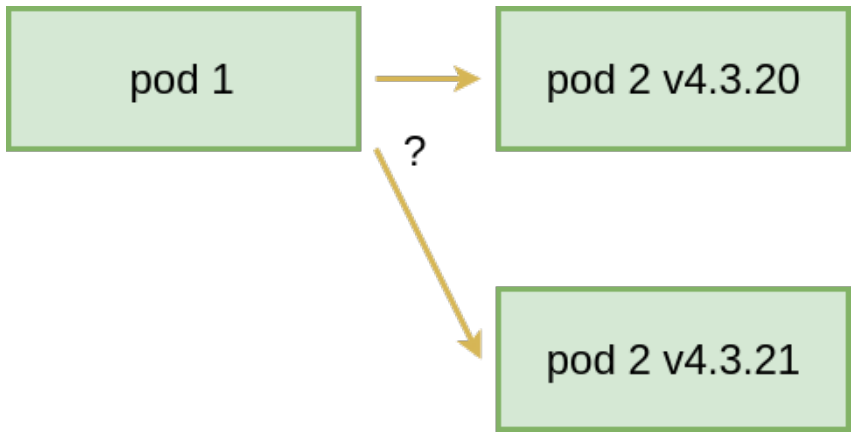
- generates secure naming information
- passes it to proxies



Limit 4

Service discovery:

- round robin load balancing mode
- no routing rules



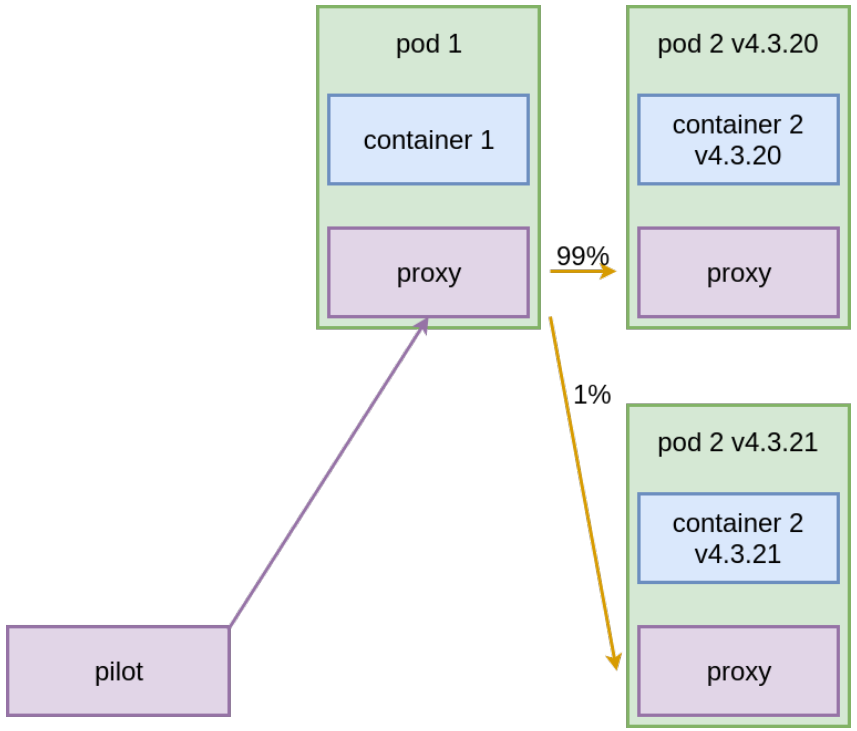
Idea 4: Traffic management

Proxies:

- get service discovery and traffic rules
- perform service discovery
- load balancing modes: round robin, random, and weighted least request

Pilot:

- manages and configures all proxy instances
- is responsible for the lifecycle of the proxies



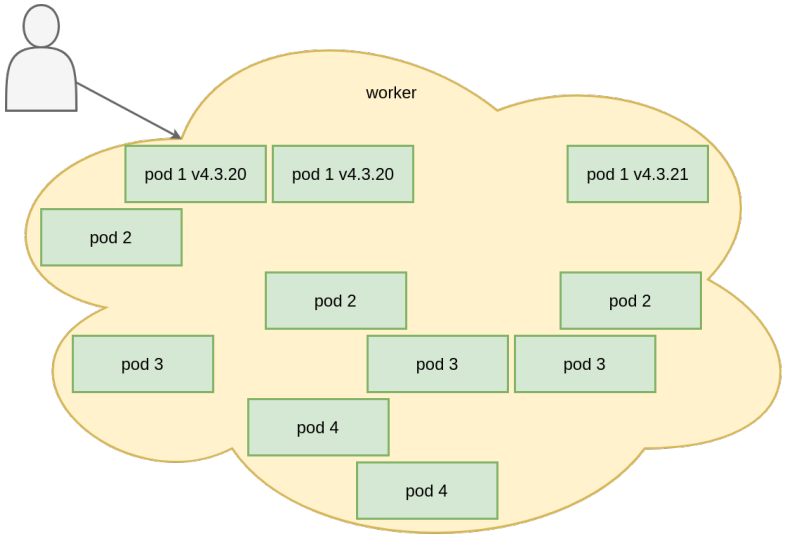
New possibilities

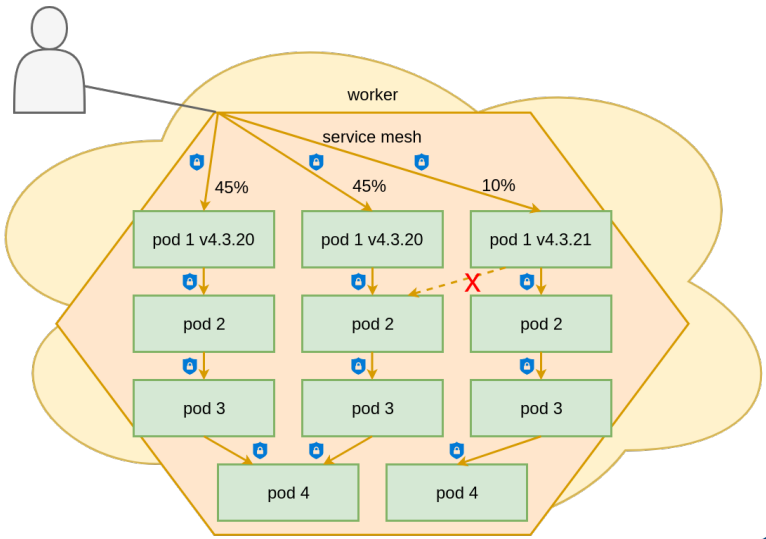
Telemetry capturing:

- collecting, querying and visualizing metrics
- collecting logs
- collecting trace spans
- visualizing the mesh

Policy enforcement:

- dynamically limit traffic to service
- modify request headers and routing
- control access to service





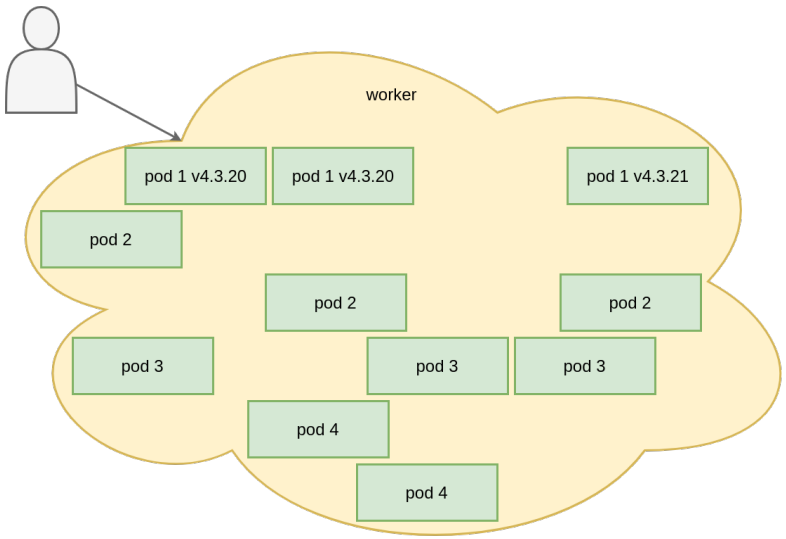
New possibilities

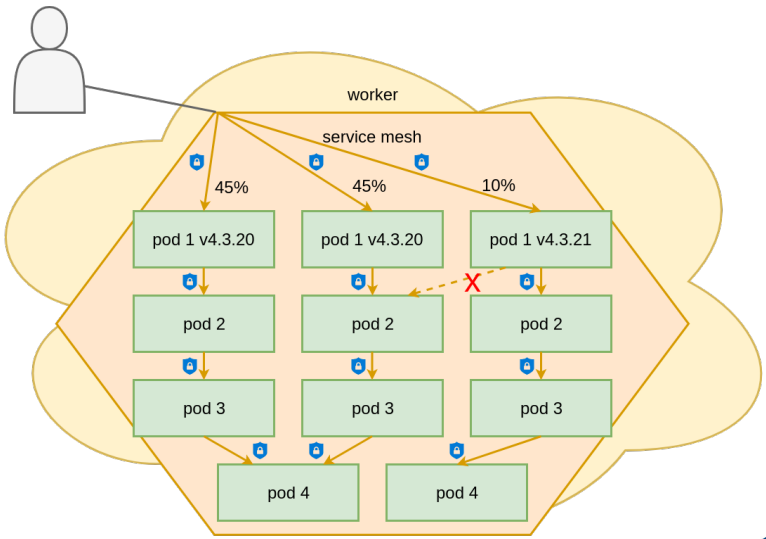
Telemetry capturing:

- collecting, querying and visualizing metrics
- collecting logs
- collecting trace spans
- visualizing the mesh

Policy enforcement:

- dynamically limit traffic to service
- modify request headers and routing
- control access to service





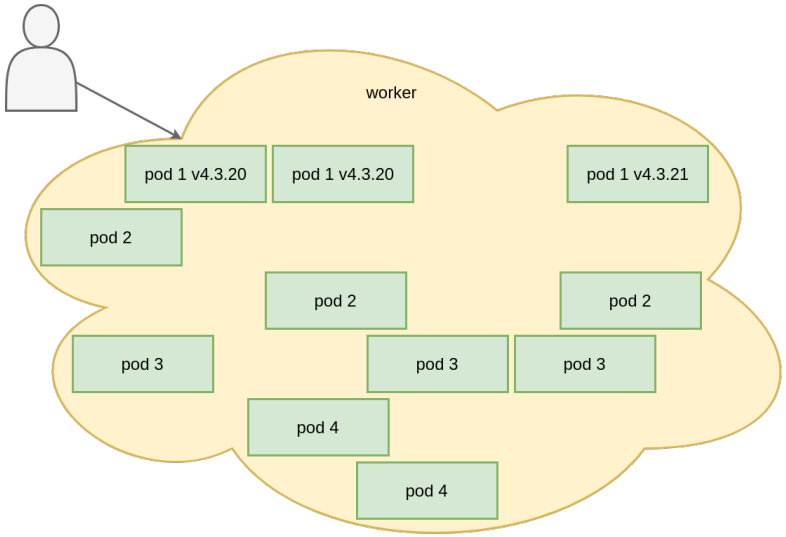
New possibilities

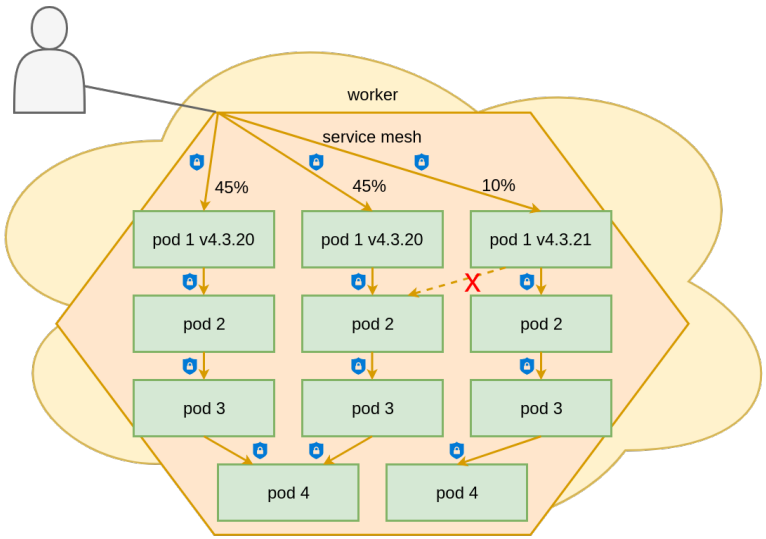
Security features:

- using authentication and authorization
- provisioning identity
- using mutual TLS

Traffic management:

- configuring request routing
- traffic shifting
- fault injection
- setting up request timeouts
- controlling ingress/egress traffic
- circuit breaking
- mirroring





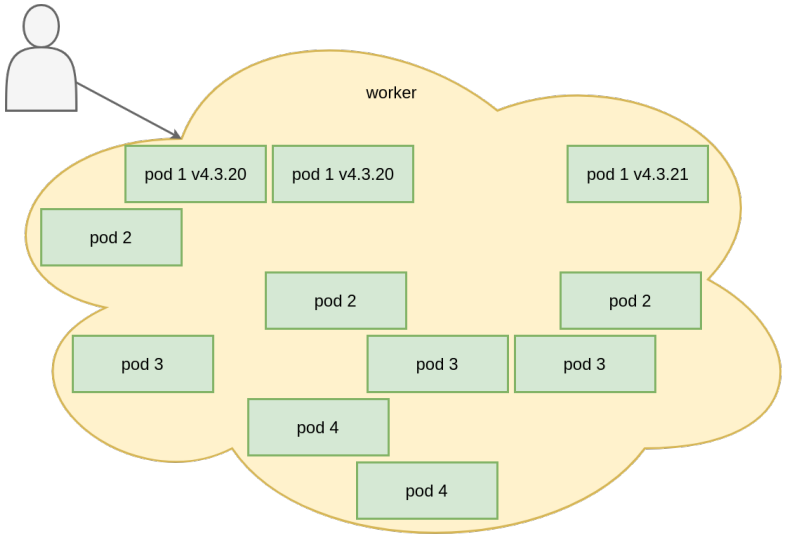
New possibilities

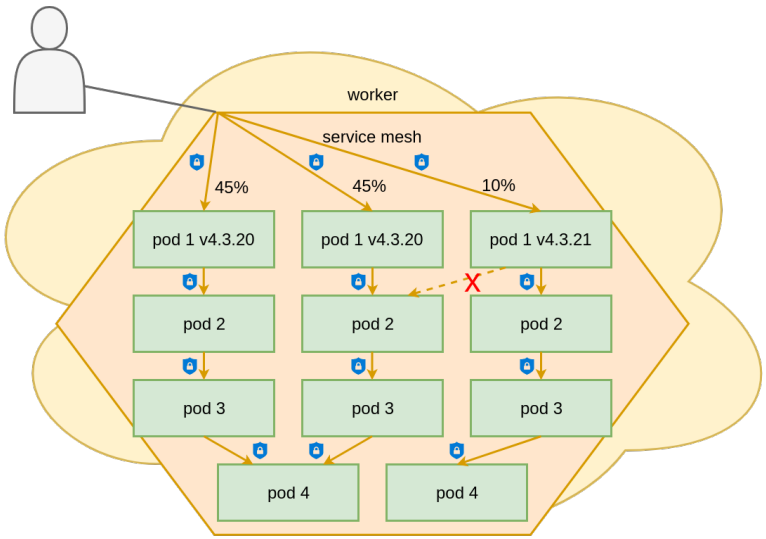
Security features:

- using authentication and authorization
- provisioning identity
- using mutual TLS

Traffic management:

- configuring request routing
- traffic shifting
- fault injection
- setting up request timeouts
- controlling ingress/egress traffic
- circuit breaking
- mirroring





Summary

Microservices/containers:

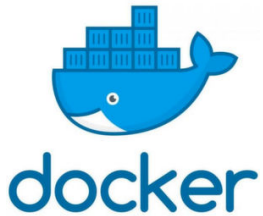
- velocity
- isolation

Scheduler/Orchestrator

- efficiency
- automation

Service Mesh

- telemetry capturing
- policy enforcement
- security features
- traffic management



Summary

Microservices/containers:

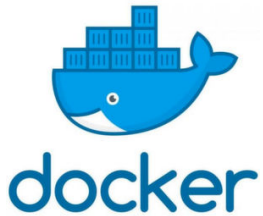
- velocity
- isolation

Scheduler/Orchestrator

- efficiency
- automation

Service Mesh

- telemetry capturing
- policy enforcement
- security features
- traffic management



Summary

Microservices/containers:

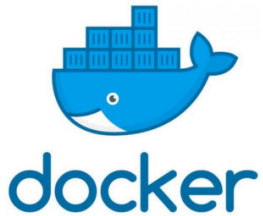
- velocity
- isolation

Scheduler/Orchestrator

- efficiency
- automation

Service Mesh

- telemetry capturing
- policy enforcement
- security features
- traffic management



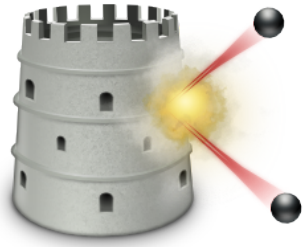
Cluster automation

Foreman:

- deployment of hosts
- on-premise and in cloud

Katello:

- lifecycle management
- errata management



Ansible:

- configuration management
- automation

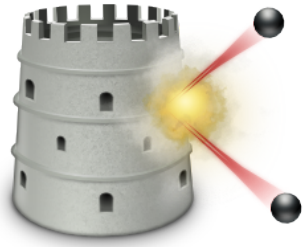
Cluster automation

Foreman:

- deployment of hosts
- on-premise and in cloud

Katello:

- lifecycle management
- errata management



Ansible:

- configuration management
- automation

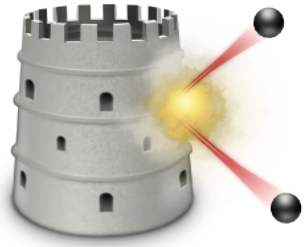
Cluster automation

Foreman:

- deployment of hosts
- on-premise and in cloud

Katello:

- lifecycle management
- errata management

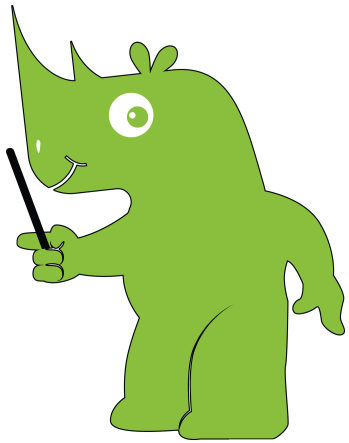


Ansible:

- configuration management
- automation

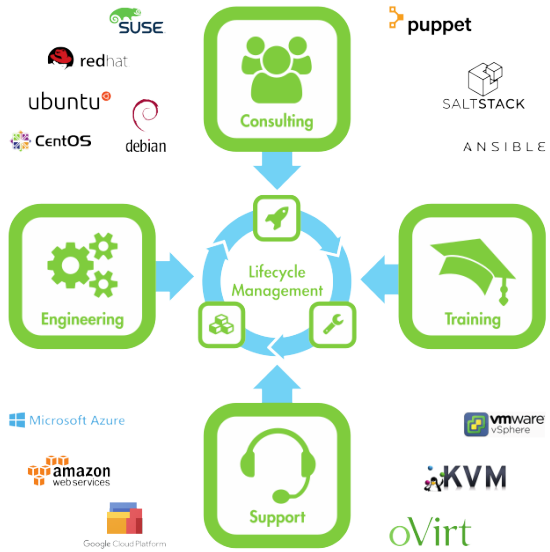
Orcharhino:

- delivery
- improvements
- quality assurance
- support
- documentation



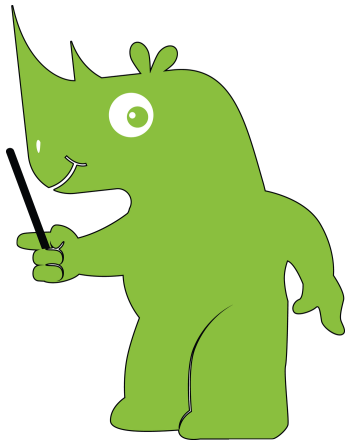
ATIX:

- consulting
- engineering
- support
- training



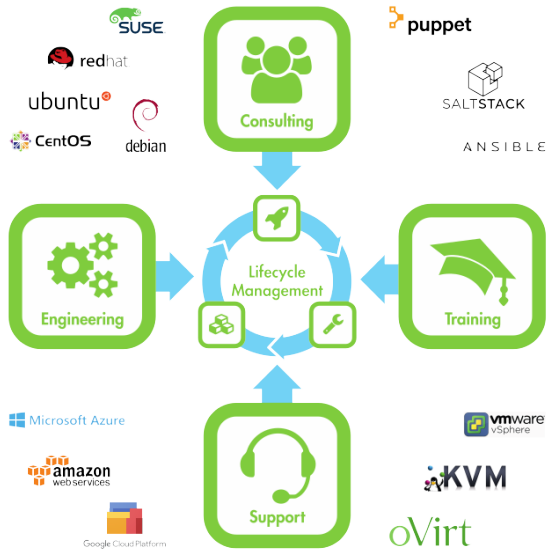
Orcharhino:

- delivery
- improvements
- quality assurance
- support
- documentation



ATIX:

- consulting
- engineering
- support
- training



ATIX booth

#atix #grazerlinuxtage2019 #container #orchestration #servicemesh

ATIX presentations

Dr. Sebastian Oehlke:

- Terraform – Ein Einblick der Möglichkeiten von Infrastructure-as-a-Code
- 4pm, i11

Manuel Bonk:

- Next Level Ansible
- 5pm, i11

ATIX presentations

Dr. Sebastian Oehlke:

- Terraform – Ein Einblick der Möglichkeiten von Infrastructure-as-a-Code
- 4pm, i11

Manuel Bonk:

- Next Level Ansible
- 5pm, i11