Zurich University of Applied Sciences

zh aw School of Engineering
InIT Institute of Applied Information Technology

CSRV - Cloud Services

Prof. Dr. Thomas M. Bohnert
Christof Marti

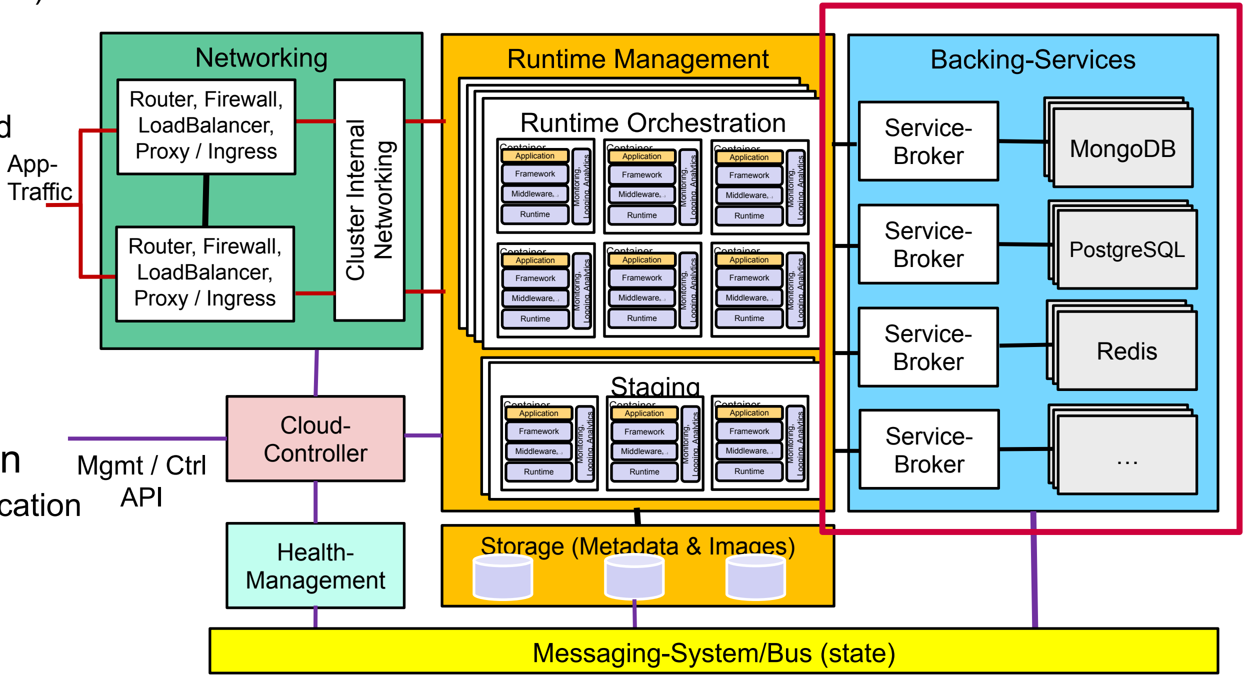
CCP2-EN Cloud Computing 2

Content

- Services
- Service Registration, Discovery and Ranking
- Service Broker Interfaces and Implementations

Recap-ARCH: Backing Services / Service Broker

- Backing Service Marketplace
  - Maintains a service catalog (marketplace) & service metadata
  - Service advertising
  - Services are either external or provided within the runtime environment
- Deployment of service instances
  - setup, credentials, …
  - Shared (multiple user accounts), Dedicated (instance per connection)
  - Access control, Single-sign on
- Bind/Unbind service to application
  - Provisioning, providing access to application
  - Configuration of application
- Service broker API
  - Possibility to add new local & 3rd party services



Definition (web/cloud service)

Endpoints:
Typically URLs, ports

Implementation:
Java? PHP? Docker? We don't know!

A service fulfils the request of a client through ***discoverable endpoints*** *of an* ***encapsulated implementation*** *described by a* ***well-defined interface*** *with a uniform* ***messaging protocol plus respective information model.***

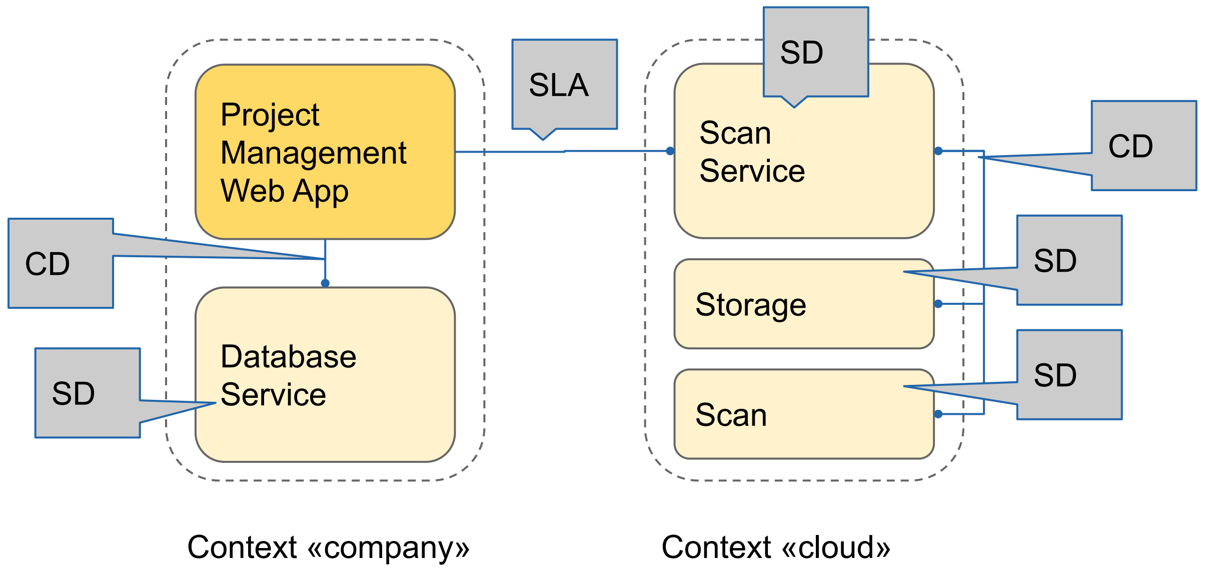
Interface:
Service description language

Protocol:
HTTP, AMQP, XMPP, ...

Application perspective:

- internal services - can be discovered and enacted with simple API calls
- external services - need more effort for signup, authentication and usage tracking

Example of application with composite service

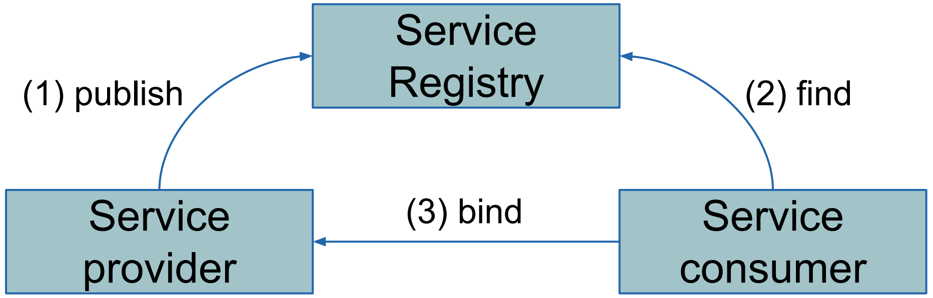


SLA: Service Level Agreement - legal terms, rights & obligations between service provider and consumer
SD: Service Description - technical terms, may include general business and legal terms
CD: Composition Description - technical binding between multiple services or application parts

Service Registry

Service Orientation: decomposition into services plus process of describing, publishing, finding, and binding services

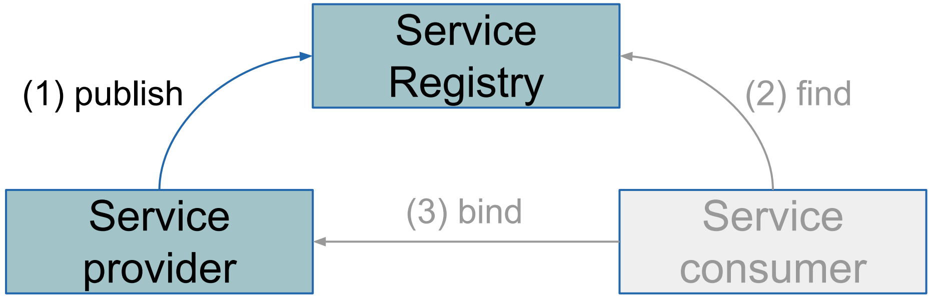
Service Registry: entity to publish and find services via their descriptions



Service Registration

Service Registration = "publish" phase

- Description + Reference to service provider/endpoint
- Description + Implementation (artifact, droplet), common in CNA (e.g. well-described containers)



Service Discovery

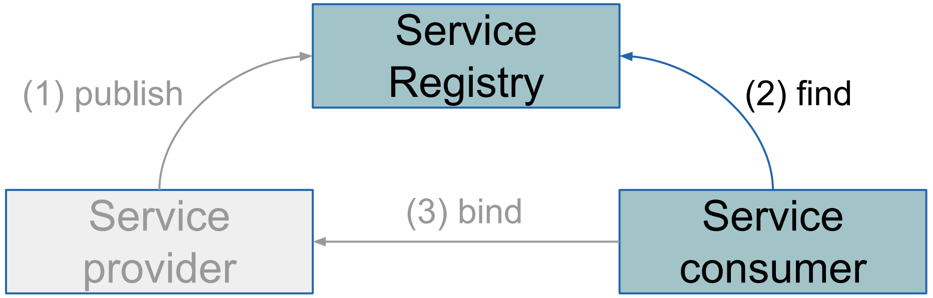
Service Discovery = "find" phase, with 2 sub-phases

Search:

- specific kind of service → functional

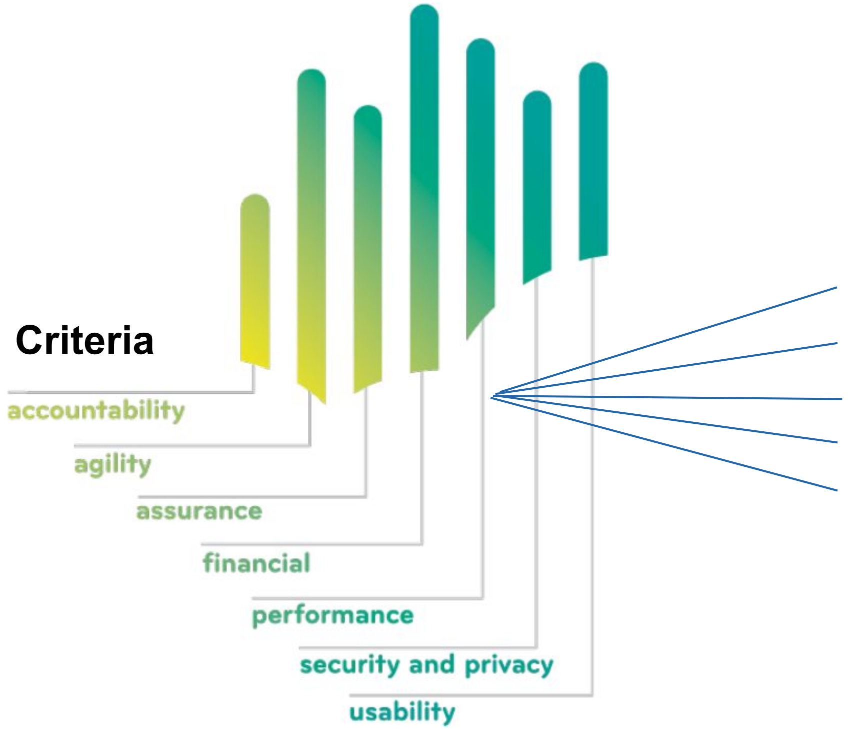
Service Ranking:

- of same or comparable functional services
- according to properties → non-functional



Service Ranking - Which one is best?

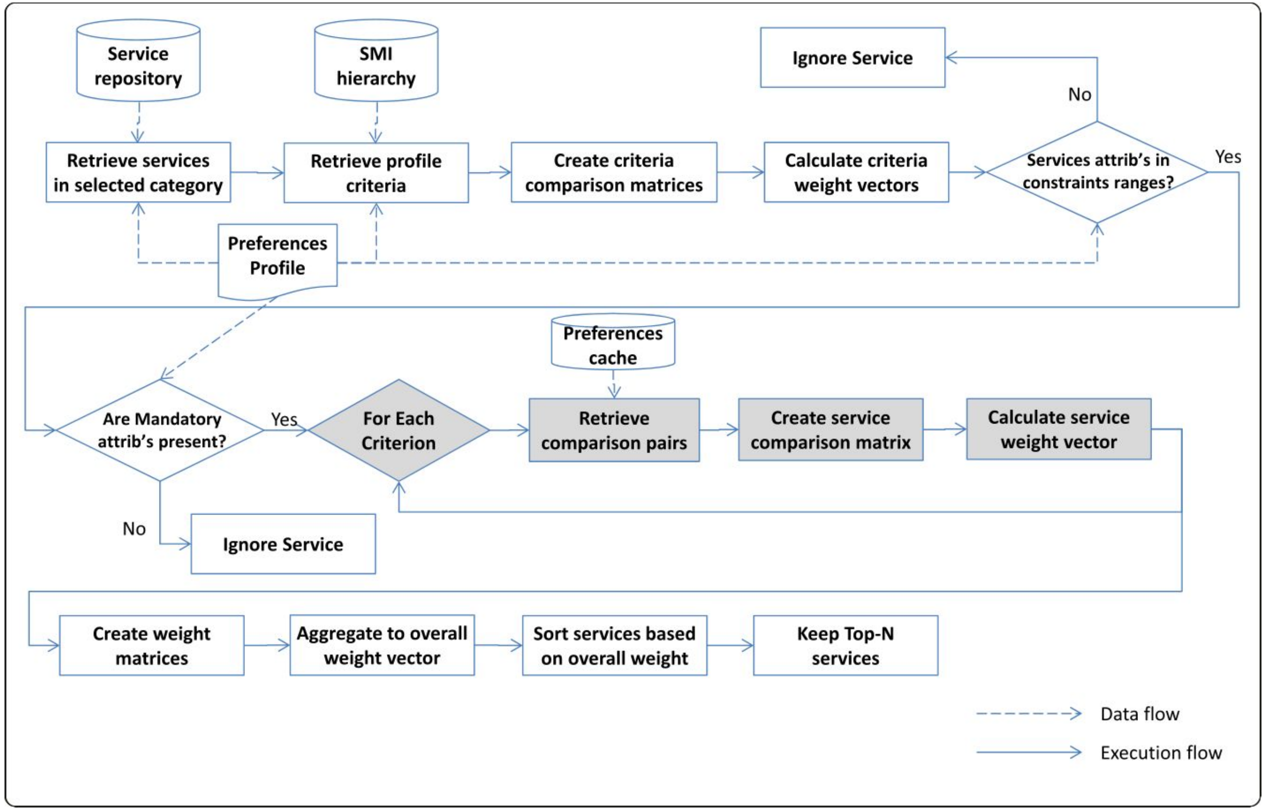
SMI: Service Measurement Index



Sub-Criteria

accuracy - adheres to requirements
functionality - set of features (discrete)
suitability - features match requirements
interoperability - interacts easily with others
response time - as indicator

Service Ranking - Flow

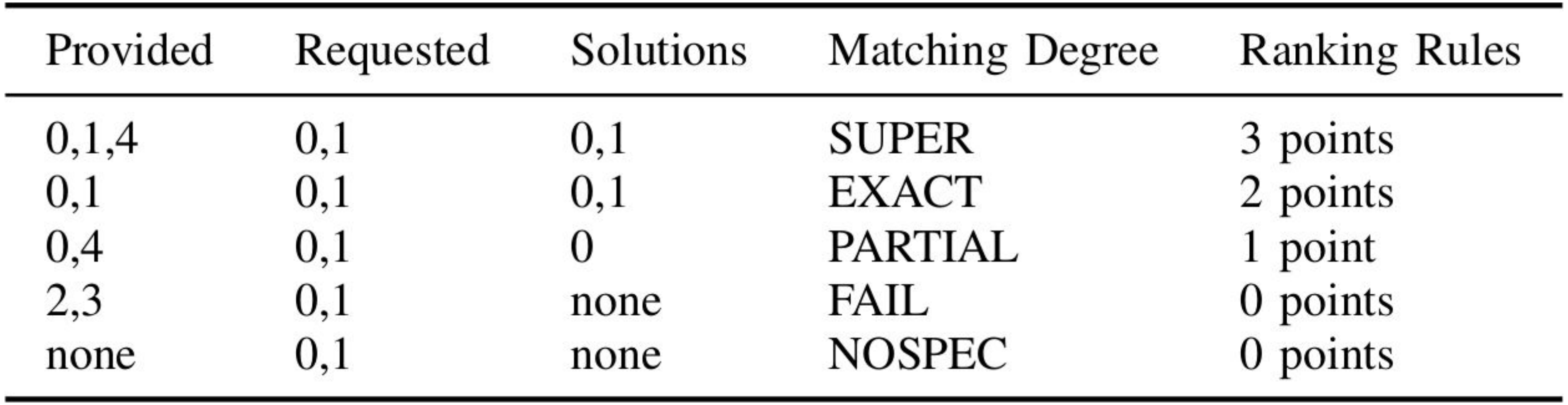

Service repository
SMI hierarchy
Ignore Service
No
Retrieve services in selected category
Retrieve profile criteria
Create criteria comparison matrices
Calculate criteria weight vectors
Services attrib's in constraints ranges?
Yes
Preferences Profile
Preferences cache
Are Mandatory attrib's present?
Yes
For Each Criterion
Retrieve comparison pairs
Create service comparison matrix
Calculate service weight vector
No
Ignore Service
Create weight matrices
Aggregate to overall weight vector
Sort services based on overall weight
Keep Top-N services
Data flow
Execution flow

[PVM15]

Service Ranking - Matchmaking

Matchmaking calculation with discrete features

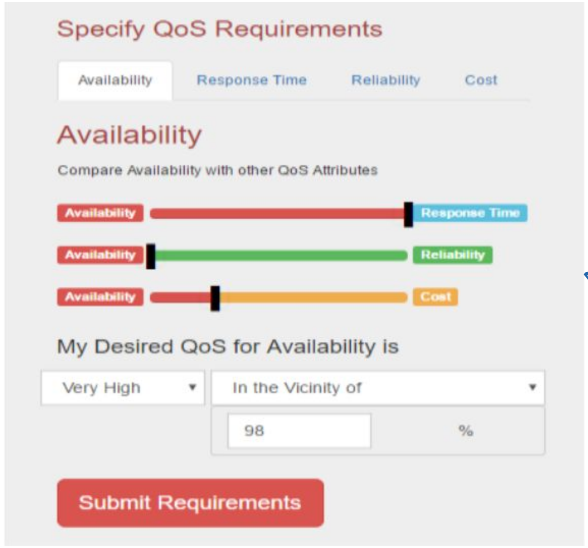
- e.g. 0 = supports SSL, 1 = is elastically scalable, 2 = zero cost



Provided	Requested	Solutions	Matching Degree	Ranking Rules
0,1,4	0,1	0,1	SUPER	3 points
0,1	0,1	0,1	EXACT	2 points
0,4	0,1	0	PARTIAL	1 point
2,3	0,1	none	FAIL	0 points
none	0,1	none	NOSPEC	0 points

[ZSK16]

Service Ranking - Input & results



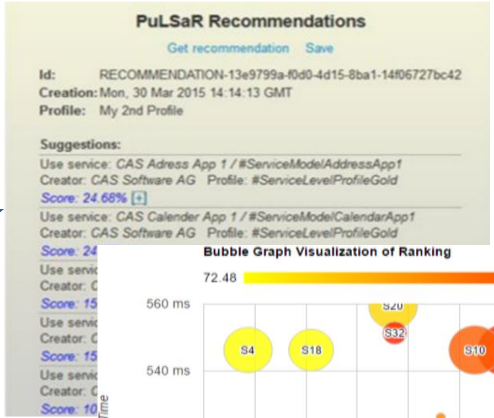
Specify QoS Requirements

Availability

Compare Availability with other QoS Attributes

My Desired QoS for Availability is

Very High | In the Vicinity of | 98 %



PuLSaR Recommendations

Id: RECOMMENDATION-13e9799a-f0d0-4d15-8ba1-14f06727bc42
Creation: Mon, 30 Mar 2015 14:14:13 GMT
Profile: My 2nd Profile

Suggestions:

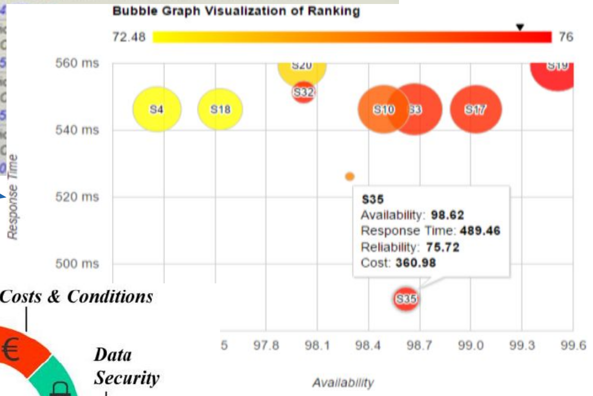
Use service: *CAS Adress App 1 / #ServiceModelAddressApp1*
Creator: *CAS Software AG* Profile: *#ServiceLevelProfileGold*
Score: 24.68% [+]

Use service: *CAS Calender App 1 / #ServiceModelCalendarApp1*
Creator: *CAS Software AG* Profile: *#ServiceLevelProfileGold*
Score: 24

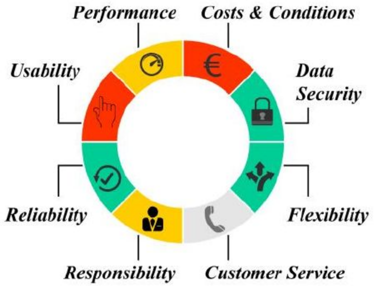
Use servi
Creator: C
Score: 15

Use servi
Creator: C
Score: 15

Use servi
Creator: C
Score: 10



S35
Availability: 98.62
Response Time: 489.46
Reliability: 75.72
Cost: 360.98



https://jisajournal.springeropen.com/articles/10.1186/s13174-015-0042-4

[TPH+14]

Service Registries in Practice

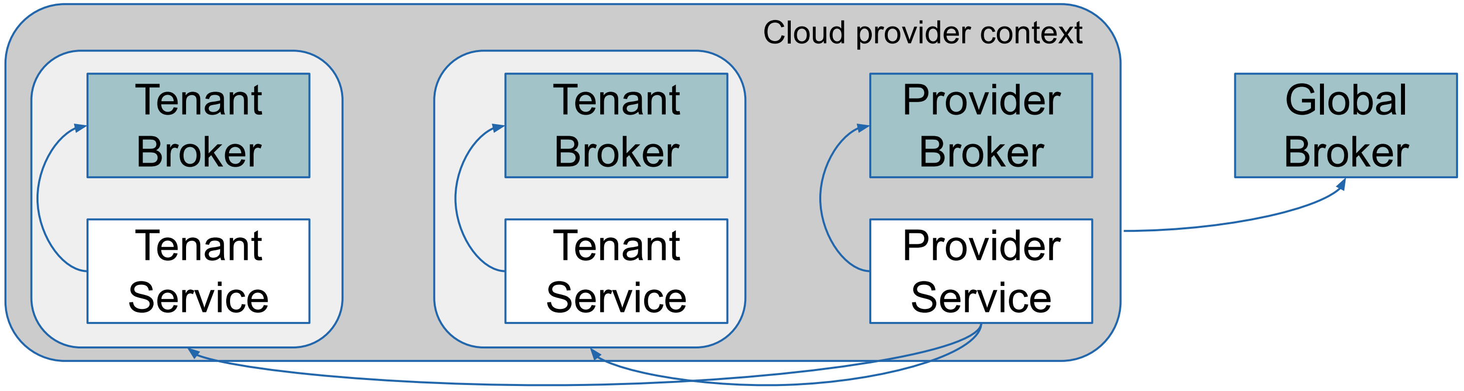
Almost never pure registry functionality

- **Broker** := Registry + service enactment (e.g. deployment, provisioning)
  → e.g. Open Service Broker API (Cloud Foundry, K8S, Openshift)
- **Repository** := Service implementation (+ Registry)
  → e.g. Docker Hub
- **Catalog** := Registry + presentation (UI)
  → e.g. Programmable Web
- **Marketplace** := Catalog + Accounting (rating, charging, billing, payment, etc)

Hybrid implementations are possible

Service Broker Categories

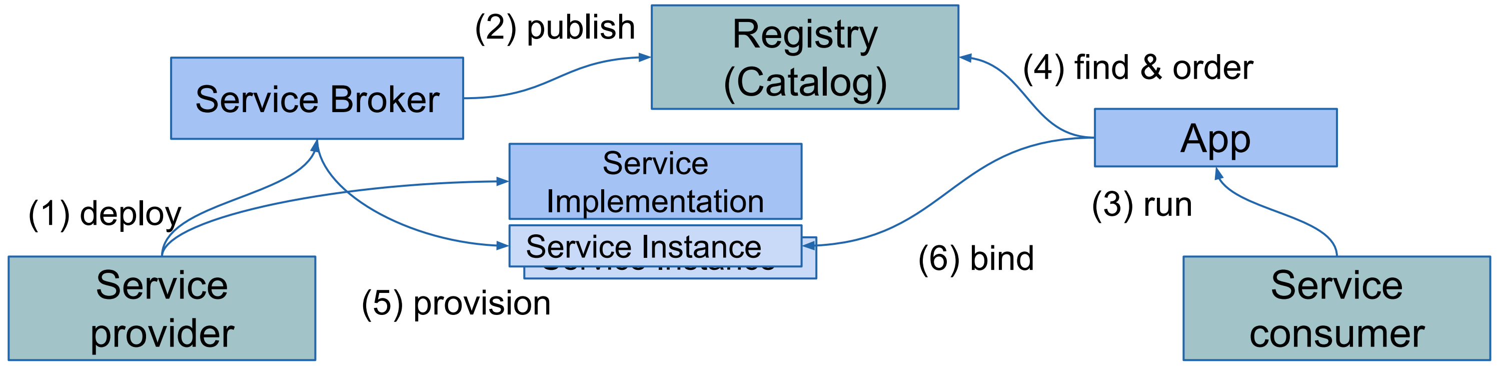
- **Global level**: provisions backing services globally across providers
- **Provider level**: contains value-added backing services within one platform
- **Tenant level**: contains shared user provided backing services



Service Brokers in PaaS

- provider managed (all-tenants) or user-provided (per-tenant)
- deploy / provision service instances for use with applications

Example for provider/tenant-level broker:

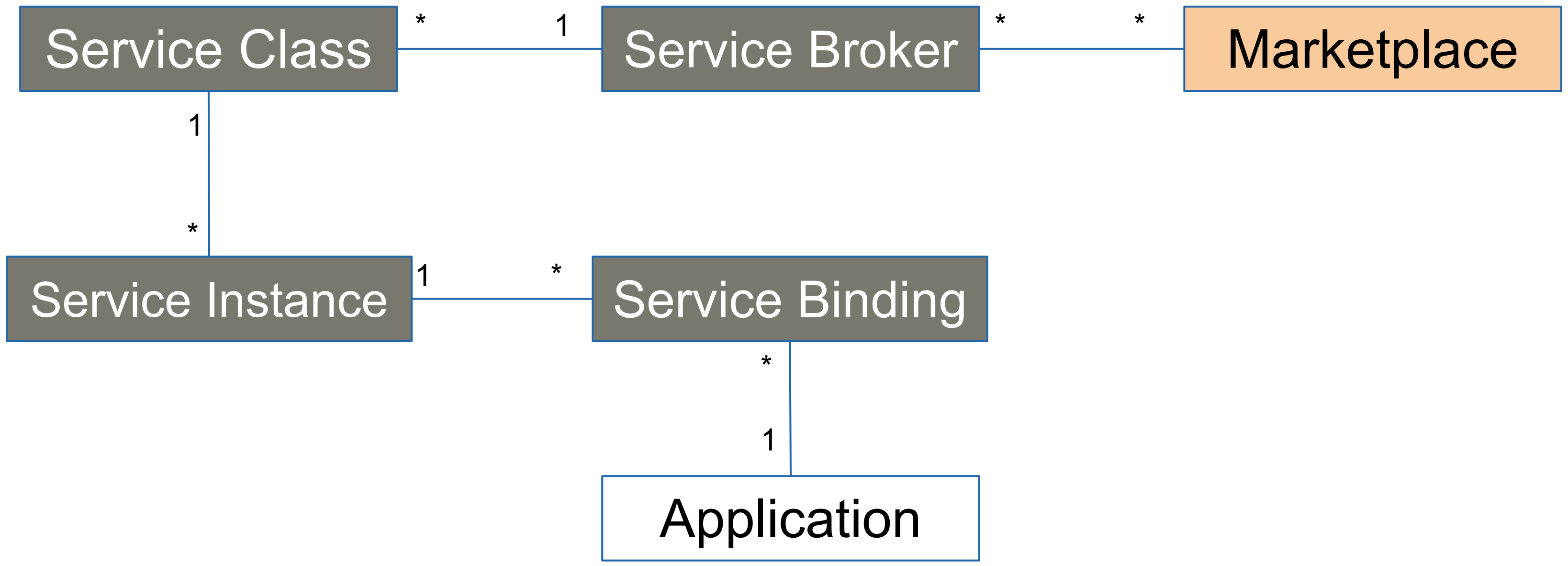


Open Service Broker API

- Standardized Interface how the Platform manages Service Instances
- **Service Broker:** Component to manage a set of backing services
  - **catalog**: provide list of of the managed service descriptions
  - **de/provision**: create/delete a Service Instances
  - **un/bind**: provide connection info to access a Service Instance
- **Important**:
  - The Service Broker only manages the Service Instances (Control-Plane)
  - The communication between Application and Service Instances is direct and is not going through the Service Broker (Data-Plane)

- Evolving industry (de-facto) standard: https://www.openservicebrokerapi.org
- Detailed Specification: https://github.com/openservicebrokerapi/servicebroker/blob/master/spec.md

Open Service Broker API - System model



Open Service Broker API - Entities

- **Marketplace:** Platform component managing the services
- **Service Broker**: manages the associated services
  - a Marketplace can registers multiple Service Brokers (usually one per type)
  - a Service Broker can be registered with multiple Marketplaces
  - Marketplace uses basic authentication to access Service Broker
- **Service Class**: Service Implementation providing the functionality
  - Large multi-tenant capable application (e.g. DB-Cluster, SaaS based service)
  - Blue-Print, image containing the runnable code (e.g. VM/Docker-Image)
- **Service Instance**: Running instance of the Service
  - Specific Tenant on a multi-tenant application (e.g. DB-Cluster, SaaS app,…)
  - Container, VM running an image
- **Service Binding**: Information to access the Service Instance
  - credentials, url, port, path, ...

OSB Functions – catalog management

- The **Catalog** endpoint returns a **list of Service Descriptions**
- The marketplace queries all registered Service Brokers to create the user facing backing-service catalog
- The Service Descriptions contains **object information** (name*, id*, description*, tags), **flags** (bindable*, plan_updatable), **metadata** (provider, documentation) and a **list of Service Plans***.

*) required fields

```
{
   "services":[
   {
      "id":"766fa866-a950-4b12-adff-c11fa4cf8fdc",
      "name":"cloudamqp",
      "description":"Managed HA RabbitMQ servers",
      "requires":[],
      "tags":[ "amqp", "rabbitmq", "messaging" ],
      "bindable": true,          // can be bound to app
      "metadata":{               // infos displayed in Catalog
         "displayName":"CloudAMQP",
         "imageUrl":"https://example.com/amqp.png",
         "longDescription":"Managed, highly available,
                            RabbitMQ clusters",
         "providerDisplayName":"84codes AB",
         "documentationUrl":"http://www..../....html",
         "supportUrl":"http://www.cloudamqp.com/support.html"
      },
      "plan_updateable": true, // can up/downgrade plan
      "plans":[...]              // see next slide
   },
   {...}
   ]
}
```

OSB Functions – catalog management

- A **Service Plan** describes a manifestation of the service regarding specific attributes
  - quantity (size, connections,
  - quality (simple, cluster, HA)
- Metadata contains
  - Feature list
  - Pricing information
  - Information presented in catalog

```
"plans":[
 {
    "id":"024f3452-67f8-40bc-a724-a20c4ea24b1c",
    "name":"bunny",
    "description":"A mid-sized plan",
    "free": "false"
    "metadata":{
        "bullets":[
            "20 GB of messages",
            "20 connections"
        ],
        "costs":[{
            "amount": { "usd":99.0 },
            "unit":"MONTHLY"
        },{
            "amount":{ "usd":0.99 },
            "unit":"1GB of messages over 20GB"
        }],
        "displayName":"Big Bunny"
    }
},
{...
}]
```

OSB Functions – service provisioning

Provision: create (deploy) and configure (provision) a new Service Instance

- Actions depend on the service type and implementation
  - instantiate a dedicated service instance (VM/container),
    e.g. deploy and provision a empty database instance or cluster
  - create an account on a multi-tenant service
    e.g. account on email-service, or an object-storage-service
  - create a new namespace on a shared service instance
    e.g. key value store, books-service (see lab)

- Some Service Brokers allow to update a Service Instance (change plan or parameter, run-time management)
- Because provisioning may take some time these requests are often asynchronous → client can/must poll for completed message.
- In CloudFoundry Services Instances are always linked to a Space

OSB Functions – service binding

Bind: make a Service Instance available to an application

- Provide connection information to access the Service Instance to the Application
- Type of Information depends on the service type
  - **credentials/secrets**: information to access the service (ideally unique for each binding / application)
  - **log-drain**: url to stream log messages to
  - **route-service**: endpoint to send network packages for processing
  - **volume-service**: mount point to access storage volume

- Not all Service Instances are bindable, some deliver value just from being provisioned (flag bindable → false)

Credentials example

```
{ "credentials": {
    "uri":
"mysql://mysqluser:pass@mysqlhost:3306/dbname",
    "username": "mysqluser",
    "password": "pass",
    "host": "mysqlhost",
    "port": 3306,
    "database": "dbname"
  }
}
```

OSB Functions – unbinding & deprovisioning

Unbind: disconnects a Service Instance from an Application

- Remove / delete connection information, inactivate credentials
- Application has no access to the service anymore

Deprovision: Delete Service Instance

- Delete Image / VM, Remove Tenant or Account, Delete Namespace
- Usually the associated data is not preserved

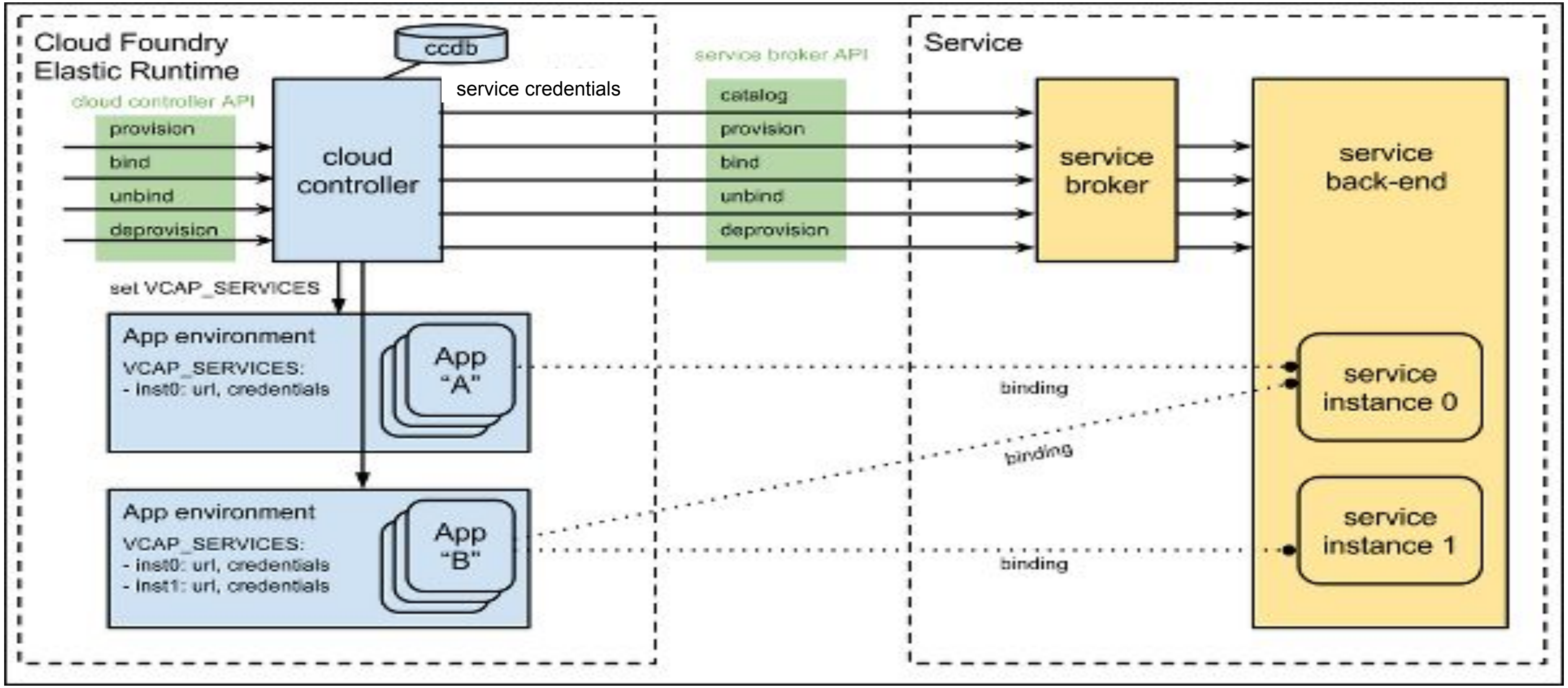
OSB API: Service Broker REST API

- List services and plans available from this broker
  GET <broker-url>/v2/catalog
- Create a new service instance (provision)
  PUT <broker-url>/v2/service_instances/:instance-id
- Create a new binding to a service instance (bind)
  PUT <broker-url>/v2/service_instances/:instance_id/service_bindings/:binding-id
- Unbind from a service instance
  DELETE <broker-url>/v2/service_instances/:instance_id/service_bindings/:binding-id
- Delete a service instance (deprovision)
  DELETE <broker-url>/v2/service_instances/:instance-id

Implementation – Cloud Foundry

- In Cloud Foundry the Cloud Controller is maintaining the Marketplace
  - Queries the Services and Plans from registered Brokers
  - Service Instances are linked to a Space and only available within it.
  - Caches Service-Bindings and injects them into the Application Environment (`VCAP_SERVICES`)
- Additional Concepts
  - **Service Key** := Service Binding without Application
    used to access a Service Instance through CLI (manage, backup, …)
  - **User-Provided-Service** := register an external Service Instances (created manually without Service Broker)

Cloud Foundry – Service Management



CloudFoundry – Service Broker Deployment Models

Cloud Foundry only requires that a **service broker implements the broker API**.

This allows multiple deployment models The following are examples of valid deployment models:

- Entire service (service implementation + broker) packaged and deployed **alongside CloudFoundry** using the same infrastructure management tools (e.g. bosh, see Platform Operation Lecture)
- Broker (and optionally service) **running as an application** in Cloud Foundry elastic runtime (this is the approach we'll take in the lab…)
- Entire service, including broker, deployed and maintained outside of Cloud Foundry by other means

CloudFoundry – Service Broker Registration

- Make the service broker known to the Cloud Controller

  `cf create service-broker <broker name> <username> <password> <brokerURI>`
  - Broker should ONLY allow access to those requestors it shared its credential (Basic Auth)
  - See: https://docs.cloudfoundry.org/services/managing-service-brokers.html#register-broker
- Requires admin role to register public Platform Service Brokers
- But normal developers can register Tenant Service Brokers
  - Only accessible within one space, but visible in organization

Example:

- Register your service broker

```
cf create-service-broker my-broker "warreng" "natedogg" https://myssvc.example.com/sb
--space-scoped
```

Required for Tenant Service Broker

Basic Auth Credentials to access Service Broker

CloudFoundry – User-Provided Service Instance

Provide information for an external Service Instances, without Service Broker

- Access to a legacy Application
- Connection to a manually managed external Database

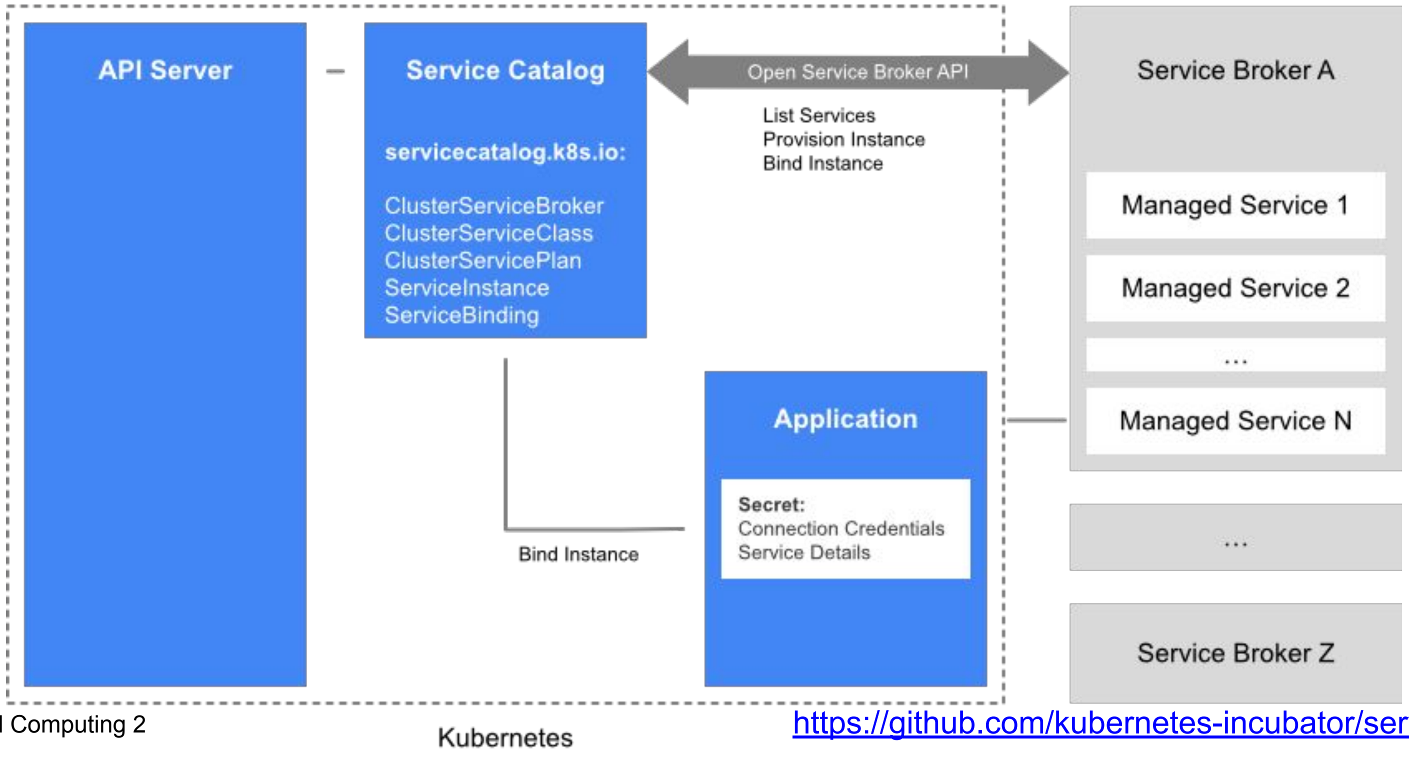
Service can then be bound to any application in the space using `cf bind`

- Provide JDBC connection info to external Mysql Database
  ```
  cf create-user-provided-service my-db -p
  '{"url":"jdbc:mysql://dublin.zhaw.ch:3306/mydb","username":"john","password":"pa55woRD"}'
  ```
- Bind it to an Application
  ```
  cf bind-service myapp my-db
  ```
- App can access it through the Environment (`VCAP_SERVICES`) in `myapp`

```
"VCAP_SERVICES": {
  "user-provided": [
  {
    "credentials": {
      "url": "jdbc:mysql://dublin.zhaw.ch:3306/mydb",
      "username": "john",
      "password": "pa55woRD"
    },
    "label": "user-provided",
    "name": "my-db",
  }]
  ...
```

Implementation: Kubernetes Service Catalog

Kubernetes incubator project “Service Catalog” implements the Service Broker API



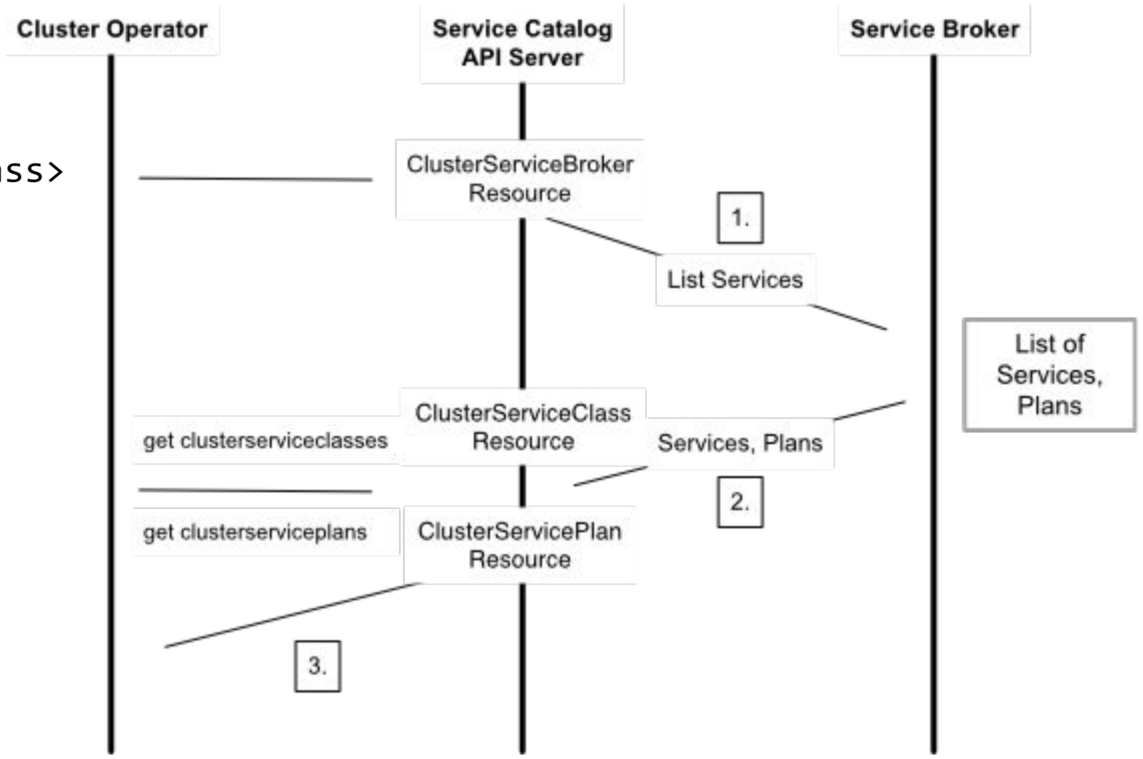
https://github.com/kubernetes-incubator/service-catalog

Kubernetes: List Service Classes

```
svcat marketplace

kubectl get clusterserviceclasses

kubectl describe class <serviceclass>
```

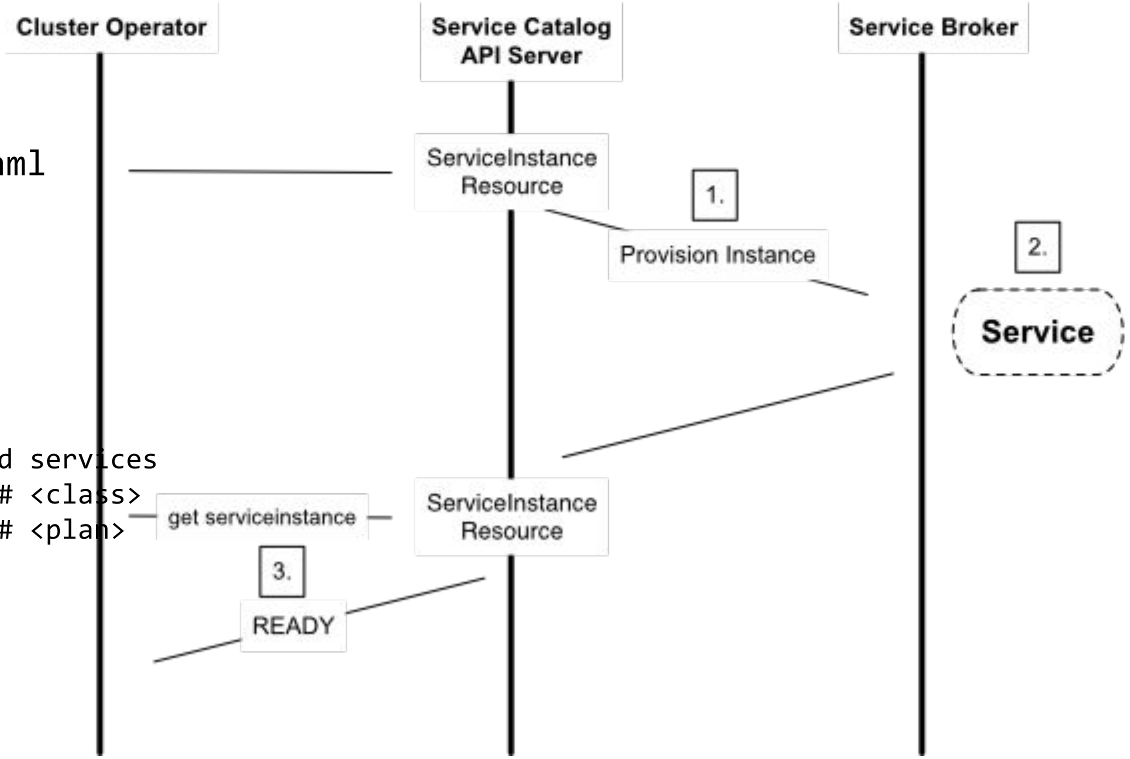

Cluster Operator
Service Catalog API Server
Service Broker
ClusterServiceBroker Resource
1.
List Services
List of Services, Plans
get clusterserviceclasses
ClusterServiceClass Resource
Services, Plans
2.
get clusterserviceplans
ClusterServicePlan Resource
3.

Kubernetes: Provision Service Instance

```
svcat provision <instance-name>
--plan <plan> --class <class>

kubectl apply -f serviceinstance.yaml

apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceInstance
metadata:
  name: pgdb          # <instance-name>
  namespace: default
spec:
  # References one of the previously returned services
  clusterServiceClassExternalName: postgres # <class>
  clusterServicePlanExternalName: 11-6-0    # <plan>
```

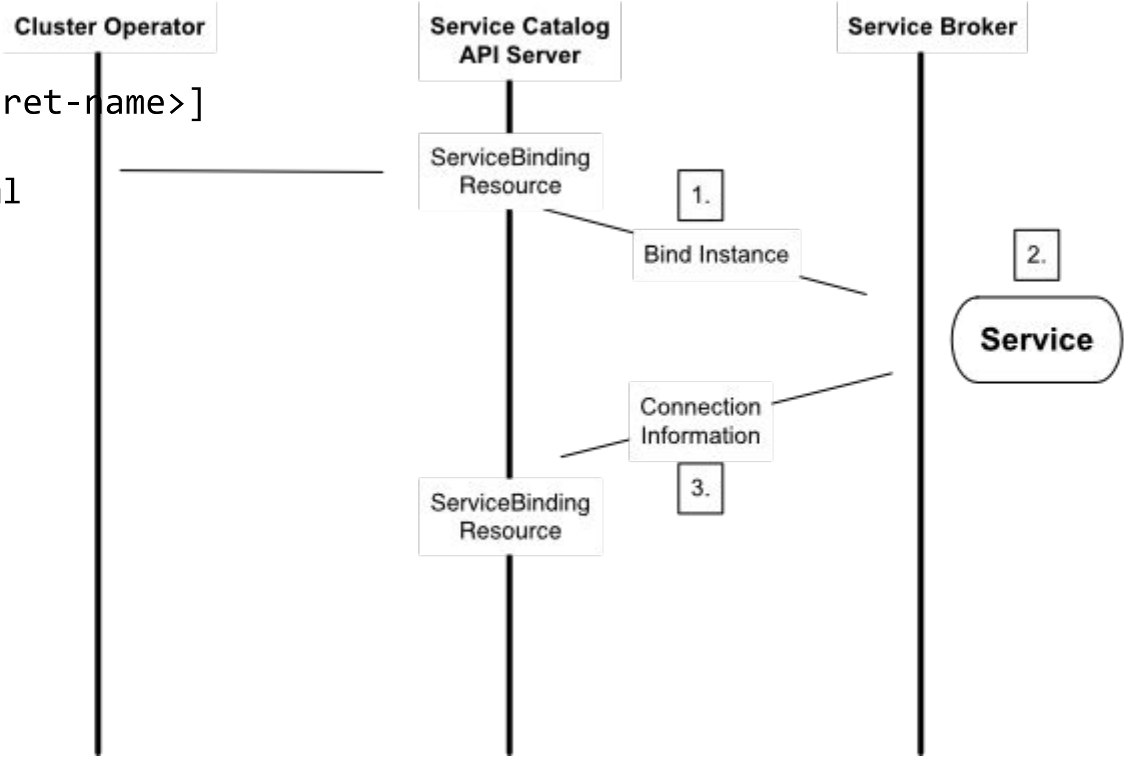


Kubernetes: Bind Service Instance

```
svcat bind <instance-name>
[--name <name>] [--secret-name <secret-name>]
```

```
kubectl apply -f servicebinding.yaml
```

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: pgdb-binding      # <name>
  namespace: default
spec:
  instanceRef:
    name: pgdb            # <instance-name>
  secretName: pgdb-secret # <secret-name>
```



Kubernetes – API usage

Service Catalog is providing the API in a kubernetes style:

- Using YAML object descriptions to create or delete a Service-...
  - **Broker** – register a service broker (URL, credentials) with the catalog

Register a broker:	`kubectl apply -f service-broker.yaml`
Get broker info:	`kubectl get brokers ups-broker -o yaml`

  - **ServiceClass** – broker is returning a list of Service Classes (descriptions)

List classes:	`kubectl get serviceclasses`
Get class details:	`kubectl get serviceclasses ups-service -o yaml`

  - **Instance** – provision a Service Instance

    kubectl apply -f service-instance.yaml
  - **Binding** – bind Service to an Application (inject connection infos)

    ```
    kubectl apply -f service-binding.yaml
    ```

    Bindings are injected through kubernetes config (e.g. using a secret)

OSB API: Managed service bindings

- List services and plans available from this broker
  REST: `GET <broker-url>/v2/catalog`
  CF: `cf marketplace`
  K8S: `svcat marketplace`
  or `kubectl get clusterserviceclasses,`
  `kubectl describe class <serviceclass>`

- Create a new service instance (provision)
  REST: `PUT <broker-url>/v2/service_instances/:instance-id`
  CF: `cf create-service <class> <plan> <instance-name>`
  K8S: `svcat provision <instance-name> --plan <plan> --class <class>`

OSB API: Managed service bindings

- Create a new binding to a service instance (bind)
  REST: PUT <broker-url>/v2/service_instances/:instance_id/service_bindings/:binding-id
  CF: `cf bind-service <app> <instance-name>`
  K8S: `svcat bind <instance-name> [--name <name>] [--secret-name <secret-name>]`
- Unbind from a service instance
  REST: DELETE <broker-url>/v2/service_instances/:instance_id/service_bindings/:binding-id
  CF: `cf unbind <app> <instance-name>`
  K8S: `svcat unbind <instance-name> [--name <name>]`
- Delete a service instance (deprovision)
  REST: DELETE <broker-url>/v2/service_instances/:instance-id
  CF: `cf delete-service <instance-name>`
  K8S: `svcat deprovision <instance-name>`

Service Brokers References

- Open Service Broker API
  - Homepage: https://www.openservicebrokerapi.org/
  - Specification: https://github.com/openservicebrokerapi/servicebroker
- Kubernetes Service Catalog
  - Homepage: http://service-catalog.drycc.cc/
  - Walkthrough: http://service-catalog.drycc.cc/docs/walkthrough/
- Service Broker CLI
  - http://service-catalog.drycc.cc/docs/cli/

Service Brokers Ecosystem

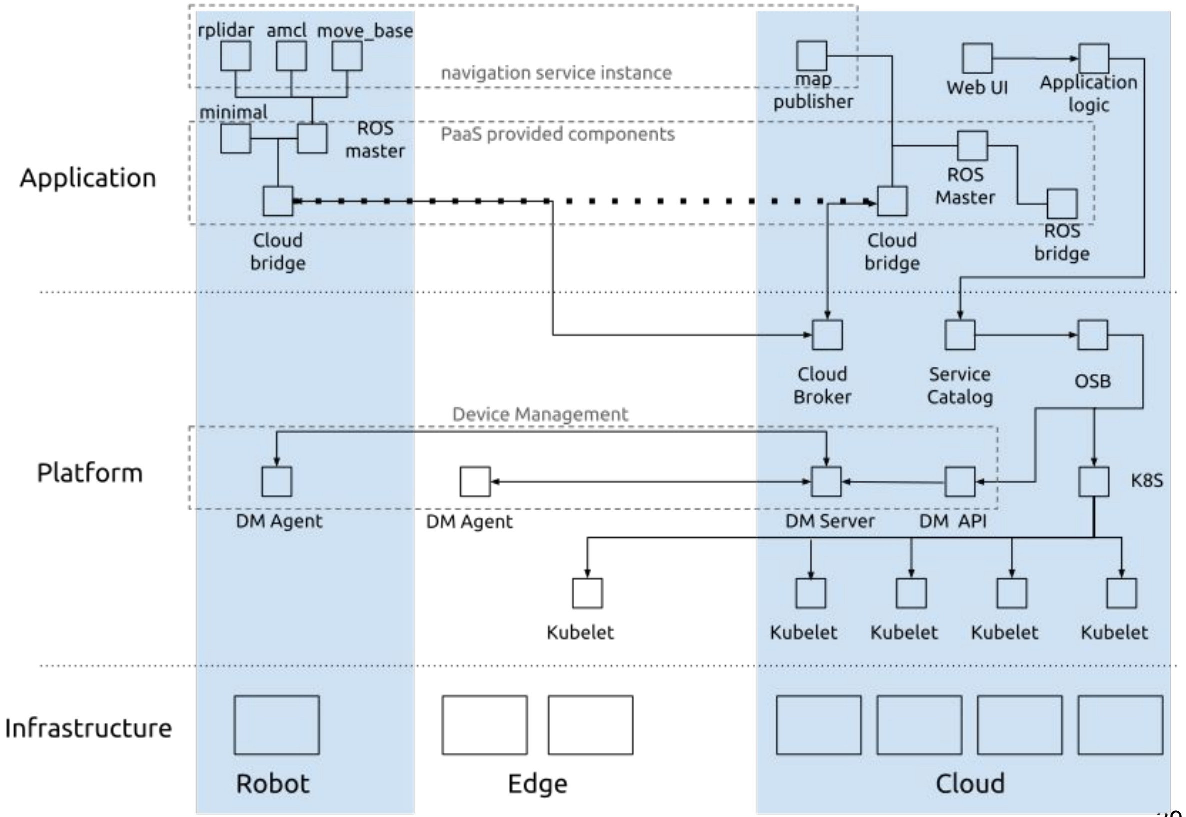
- Cloud Foundry Community Service Brokers
  - https://github.com/cloudfoundry-community?q=service%20broker
    Community provided for S3, RDS, PostgreSQL, etcd, elasticsearch, ...
- Google Cloud Platform Service Brokers
  - https://docs.pivotal.io/partners/gcp-sb/
    e.g Cloud Storage, Bigtable, BigQuery, PubSub, Cloud SQL, Machine Learning, Spanner, Stackdriver (analytics tool)
- Microsoft Azure Service Brokers
  - https://github.com/Azure/meta-azure-service-broker
    e.g. Storage, Redis Cache, DocumentDB, Service Bus & Event Hub, SQL DB, Key Vault
- AWS Service Brokers
  - https://docs.pivotal.io/aws-services
    e.g. S3, Aurora, DynamoDB, RDS (MySQL, MariaDB, Oracle, SQL-Server), SQS

Using service brokers for a robotic PaaS on K8S

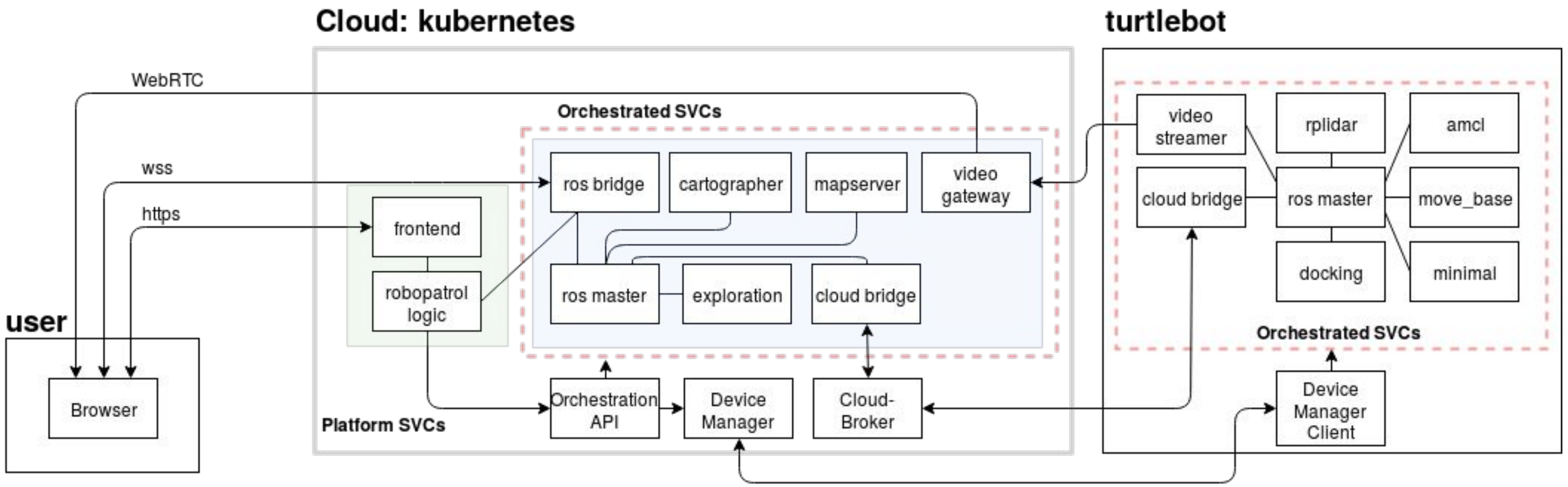
"OSB" in figure is implemented by two different brokers:

- "helm broker": uses Helm to deploy "charts" on K8S nodes
- "DM-broker": uses Device Manager service to spawn processes on robots

Applications can use the K8S API directly to request service instances



RoboPatrol application components


Cloud: kubernetes
turtlebot
user
WebRTC
wss
https
Orchestrated SVCs
ros bridge
cartographer
mapserver
video gateway
frontend
robopatrol logic
ros master
exploration
cloud bridge
Browser
Orchestration API
Device Manager
Cloud-Broker
Platform SVCs
video streamer
rplidar
amcl
cloud bridge
ros master
move_base
docking
minimal
Orchestrated SVCs
Device Manager Client