Deliver Your Cloud Native Application with Design Pattern as Code

Rintaro Sekino, Jun Makishi
NTT Communications

How do you deal with Infrastructure?

Is your Infrastructure as Code **well organized** to make changes easily?

Do you **share** Infrastructure as Code across multiple teams?

Do you **spend little time** on Infrastructure as Code implementation compared to application business logic?

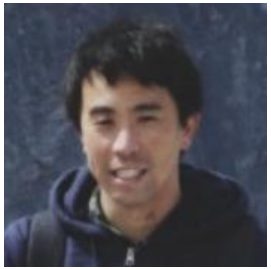
Takeaway

Infrastructure as Code (IaC) must solve **consistency and reusability** for infrastructure **Manifest** and its delivery **Pipeline**

Our answer: ***Design Pattern as Code***

- Derived from our IaC experience in production.
- Is powered by Cuelang and Tekton.

Speakers



Jun Makishi

Senior Architect, NTT Communications

@JunMakishi

https://github.com/j-maxi

Rintaro Sekino

Site Reliability Engineer, NTT Communications

@TAR_O_RIN

https://github.com/sekinet

Our Company

NTT Communications: Global ICT Service Provider

Network Service Coverage

190+

countries/regions

Total capacity of global cable systems

16.6 Tbps

Datacenter

20+

countries/regions

Server floor space

400k m²+

Cloud service

15

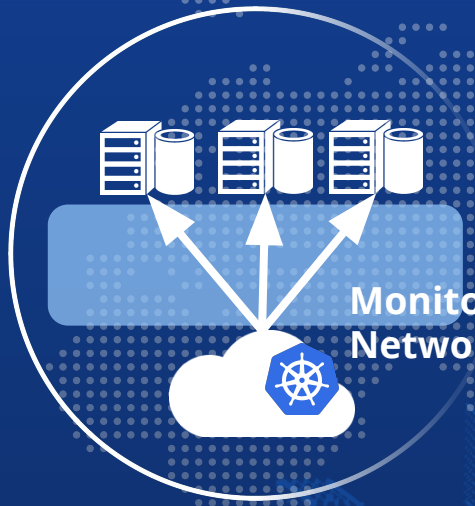
countries/regions

DC with cloud

35

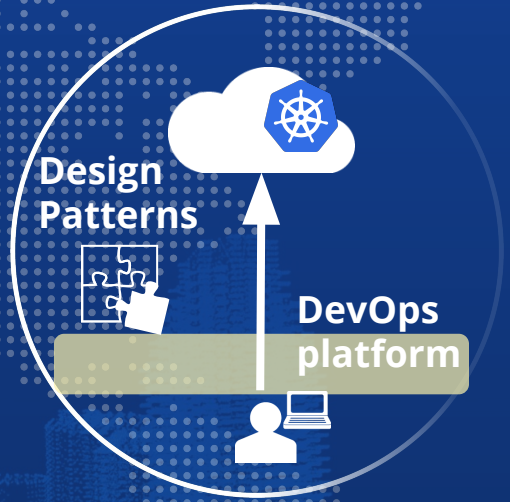
locations

Examples of Our Cloud Native Journey



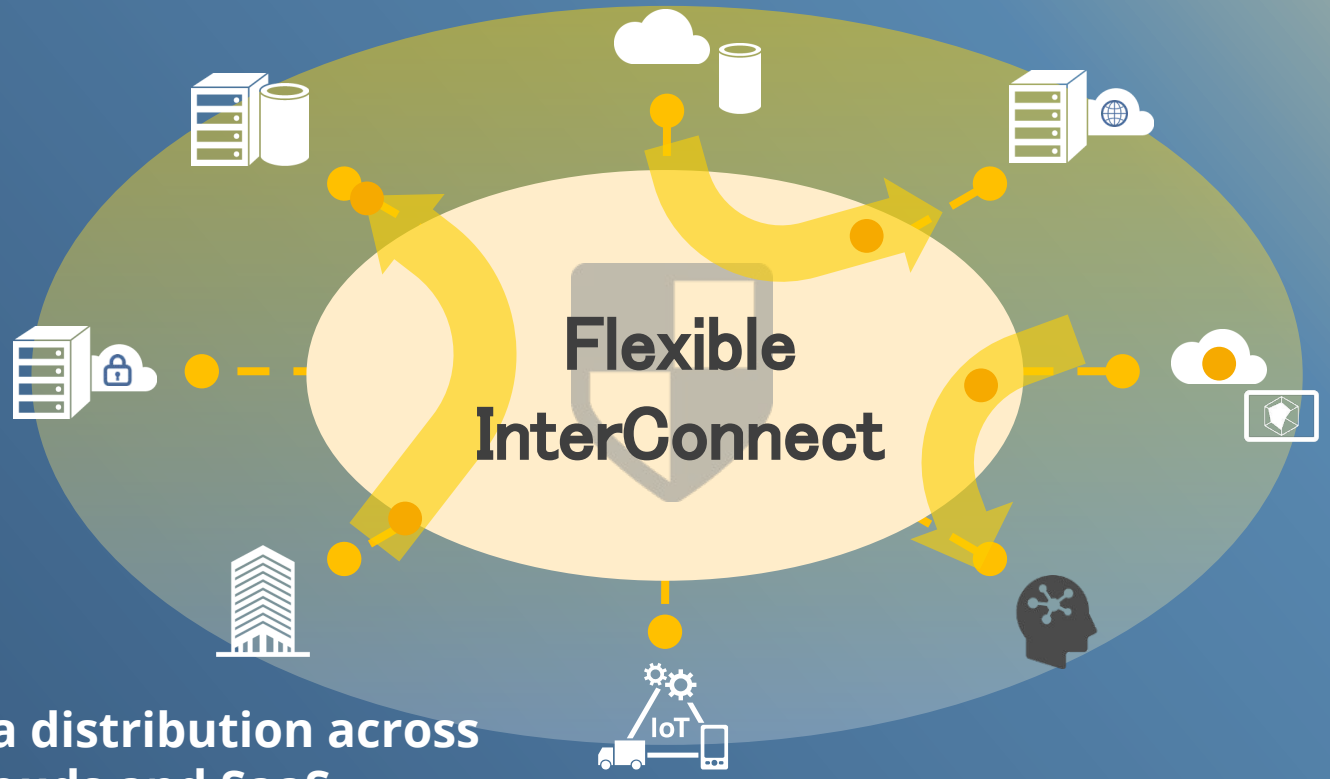
Operation team to apply modern Cloud architecture to monitor network service equipments

SRE team to manage containerized application and its release pipeline



SRE team to support end-to-end application development cycle

Flexible InterConnect



Secure data distribution across multiple clouds and SaaS

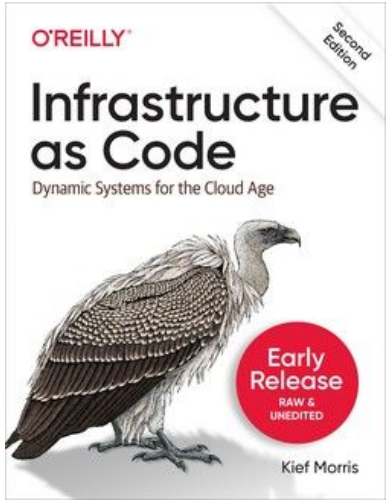
Our challenges in Infrastructure as Code

Codify everything for Infrastructure

Define everything as code.

Defining all your stuff "as code" is a core practice for making changes rapidly and reliably. There are a few reasons why this helps:

- *Reusability*
- *Consistency*
- *Transparency*



https://www.oreilly.com/library/view/infrastructure-as-code/9781098114664/

Chaos of code

Complicated and fragile code.

- Many project-specific scripts were baked in.
  - Hard to share.
- Parameter hell.
  - envsubst, Jinja, etc.
- Tangled code.
  - Dependencies between different tools like Terraform, Kubernetes, Pipeline, etc.

We don’t see *Reusability* and *Consistency*.
Did we lack discipline?

Rethink Infrastructure as Code

What we wanted:

- Software approach to share code: abstract, module, group…
- **Unified interface** for every infrastructure provider.
- Take **pipeline/operation** into account from the design.

⬇

Design Pattern as Code

Reusable and **composable** pattern of Cloud Native architecture, written in a powerful language for a software engineer.

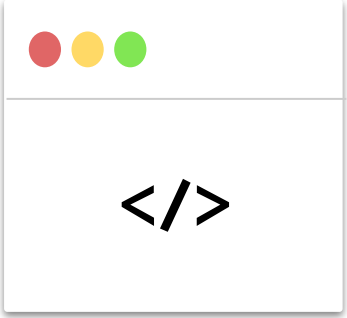
What is Design Pattern as Code

Design Pattern as Code addresses the following problems:

- How to deliver a Cloud Native application
- How to integrate Cloud Native solutions

Design Pattern

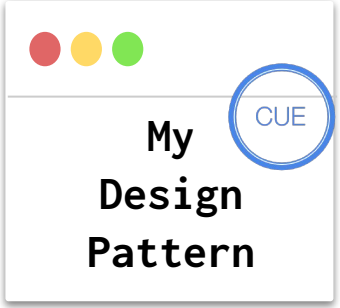
- Declares all of the infrastructure provider Manifest,
- Puts Manifest and Pipeline together, and
- Is **composable** with other Design Patterns.



Design Pattern schema

Design Pattern written in Cuelang, consists of

- Infrastructure Manifest
- Delivery Pipeline




```
resources: {
  kubernetes: {
    deployment: ...
    service: ...
  }
  gcp: {
    monitoring: ...
  }
}
```

Infrastructure Manifest for all providers

```
tasks: {
  build: ...
  deploy: ...
  test: …
}
```

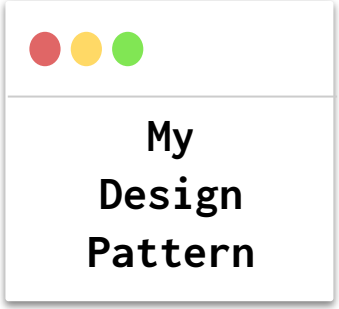
Delivery Pipeline

Tekton for workflow engine

Tekton - Declarative Pipeline to run the tasks.

- Each task is isolated from others.
- Can compose a new task to pipeline easily.

Design Pattern generates Tekton Pipeline from task declarations.




```
tasks: {
  build: ...
  deploy: ...
  test: …
}
```

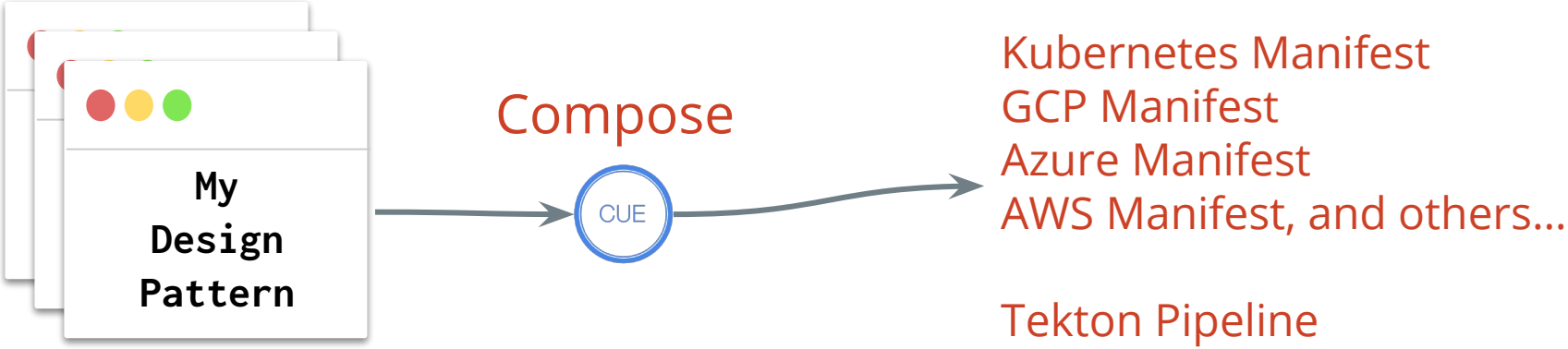
Delivery Pipeline

```
kind: Pipeline
apiVersion: tekton.dev/v1beta1
spec:
  tasks:
  - name: build
  - name: deploy
  - name: test
```

Cuelang for unified interface

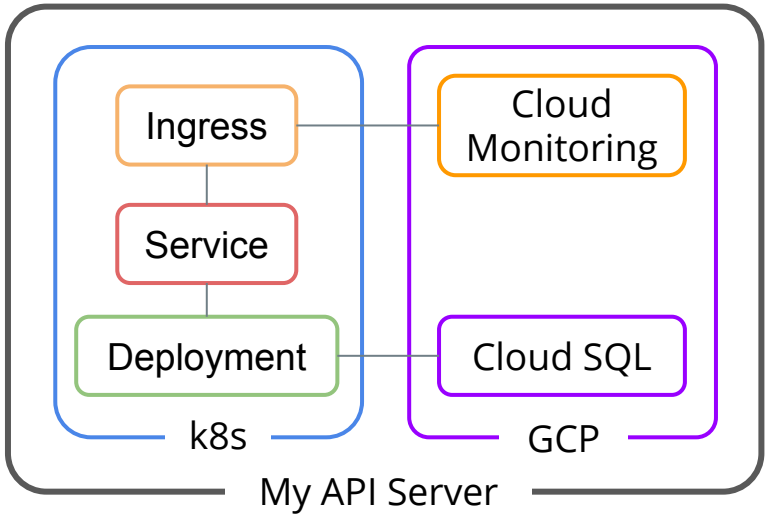
Cuelang to express Design Pattern as Code.

- Powerful typed language focusing on declaring data.
- Designed for scale, generating configs from multiple patterns.
  - Commutative and idempotent to always gives us the same results.

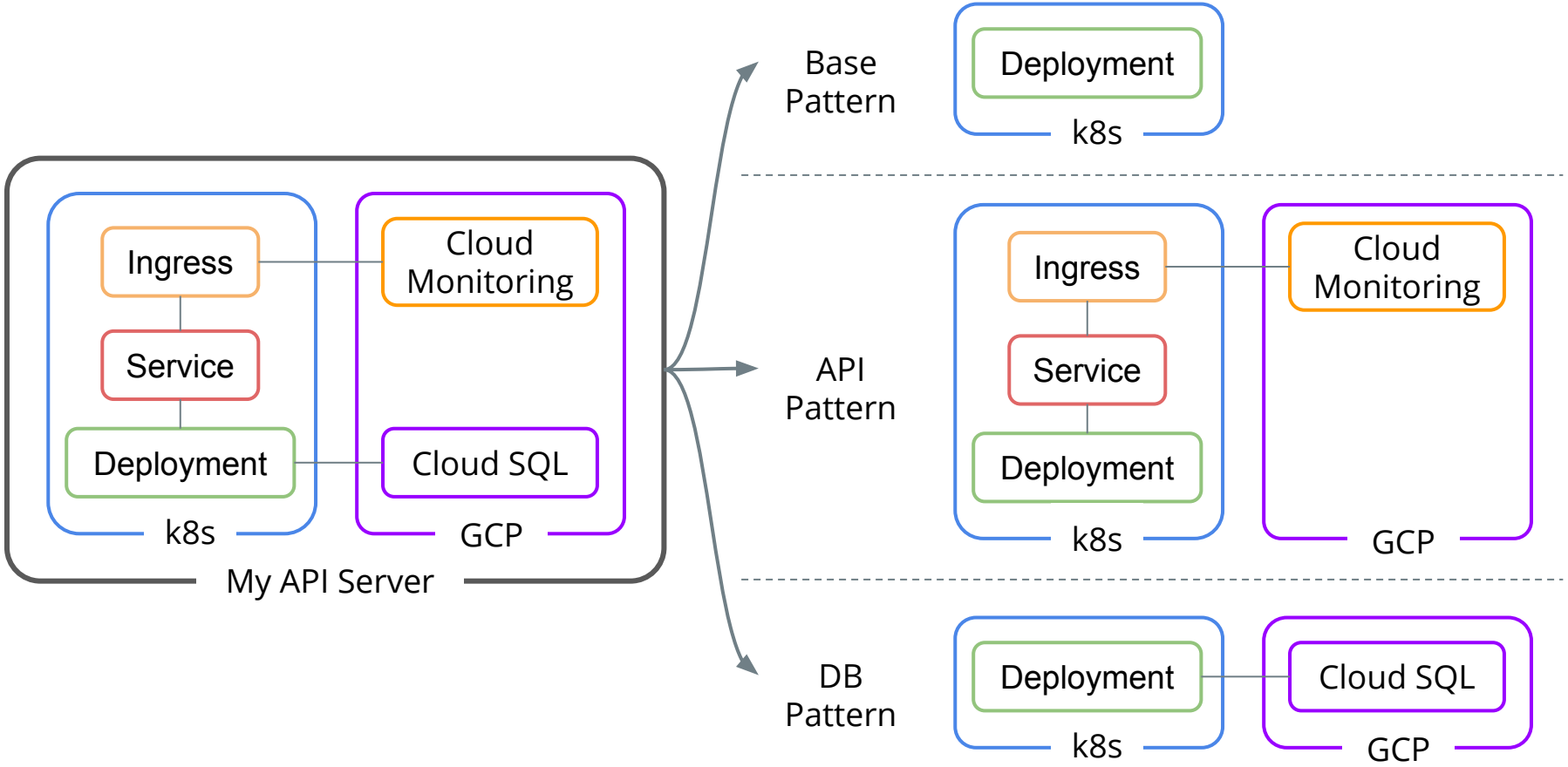


Example of Design Pattern as Code

Break Manifest to reusable portions

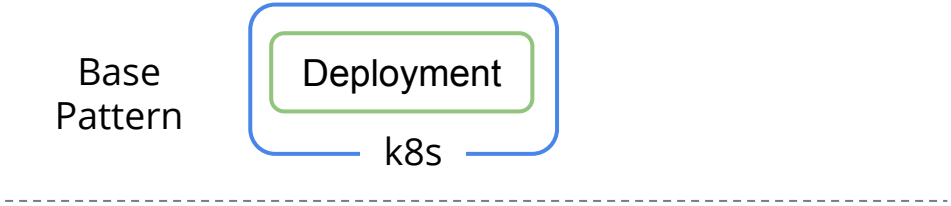


Break Manifest to reusable portions



Compose patterns

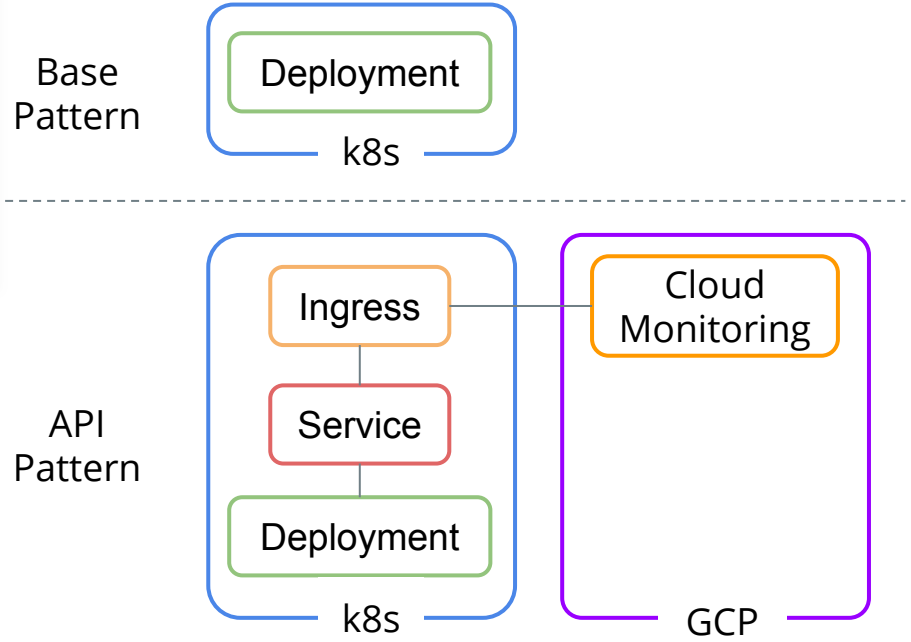
```yaml
kind: Deployment
apiVersion: v1
spec:
  templates:
    spec:
      containers:
      - name: sample
        image: sample:1.0.0
```



Compose patterns

```yaml
kind: Ingress
...
kind: Service
...

kind: Deployment
apiVersion: v1
metadata:
  labels:
    app: sample
spec:
  templates:
    spec:
      containers:
      - name: sample
        image: sample:1.0.0
        ports:
        - containerPort: 8080
```

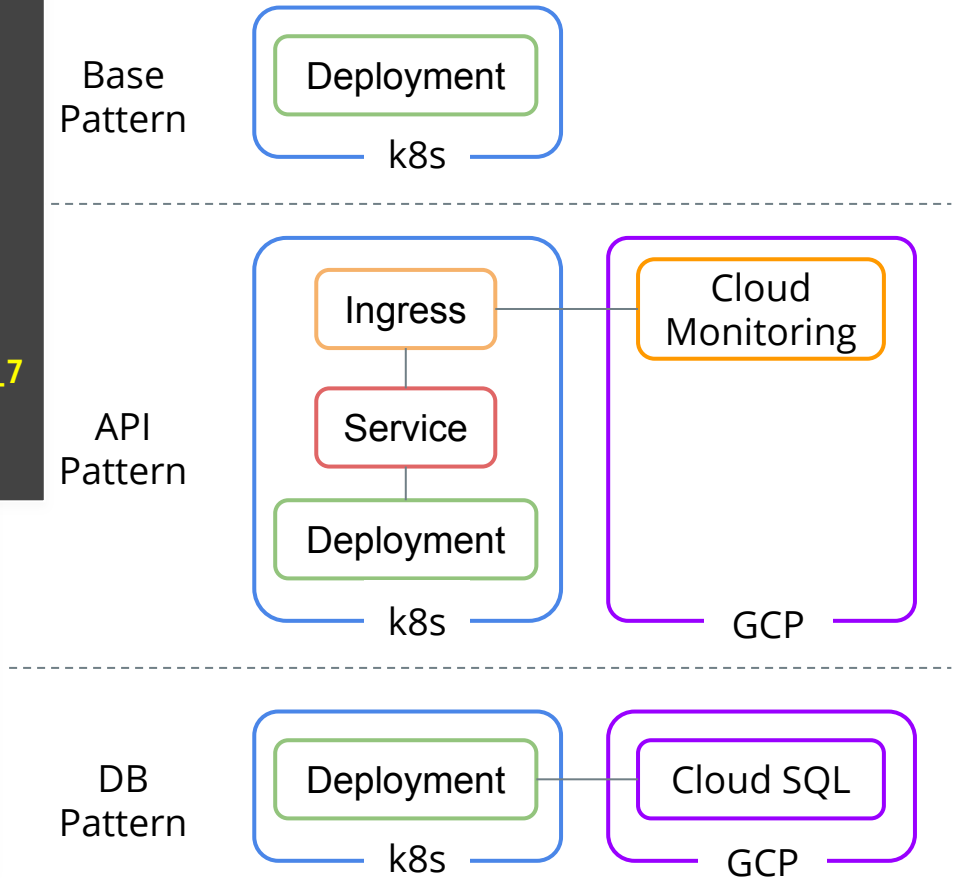
```yaml
uptimeCheckConfig:
  period: 300s
  monitoredResource:
    type: uptime_url
    labels:
      host: example.com
```



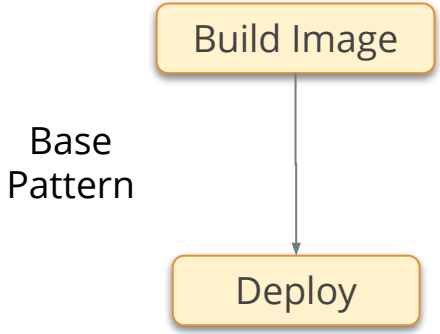
Compose patterns

```
kind: Ingress
...
kind: Service
...

kind: Deployment
apiVersion: v1
metadata:
  labels:
    app: sample
spec:
  templates:
    spec:
      containers:
      - name: sample
        image: sample:1.0.0
        ports:
        - containerPort: 8080
      - name: cloudsql-proxy
```

```
uptimeCheckConfig:
  period: 300s
  monitoredResource:
    type: uptime_url
    labels:
      host: example.com

cloudSQLInstance:
  databaseVersion: MYSQL_5_7
  settings:
    tier: db-n1-standard-1
```



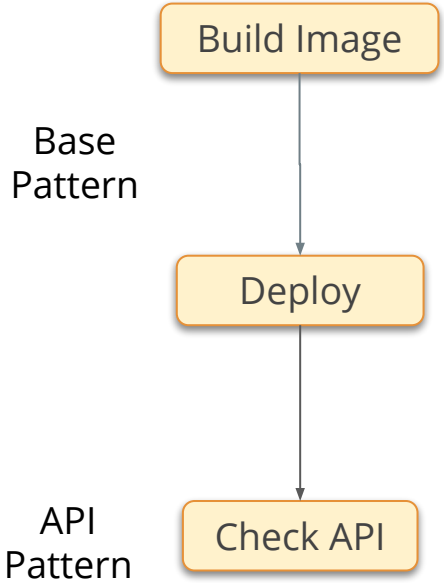
Compose pipeline tasks




```yaml
kind: Pipeline
apiVersion: tekton.dev/v1beta1
spec:
  tasks:
  - name: build-image
    steps:
    - name: run-buildkit
  - name: deploy
    steps:
    - name: generate-manifest
    - name: kubectl-deploy
    runAfter:
    - build-image
```

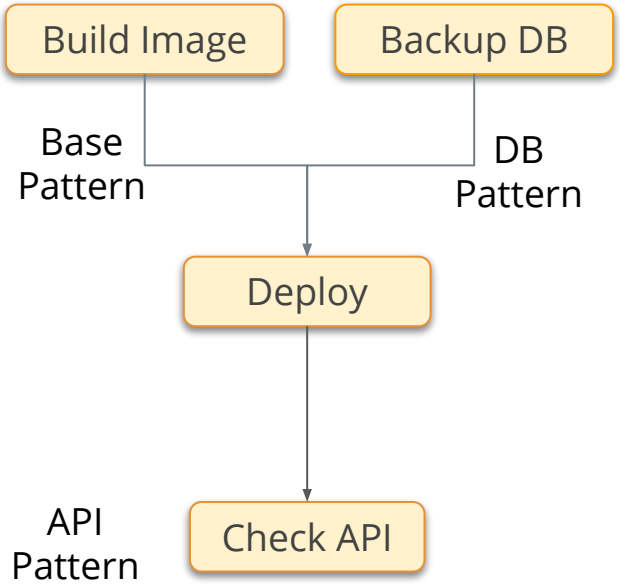
Task to compose patterns for Manifest

Compose pipeline tasks



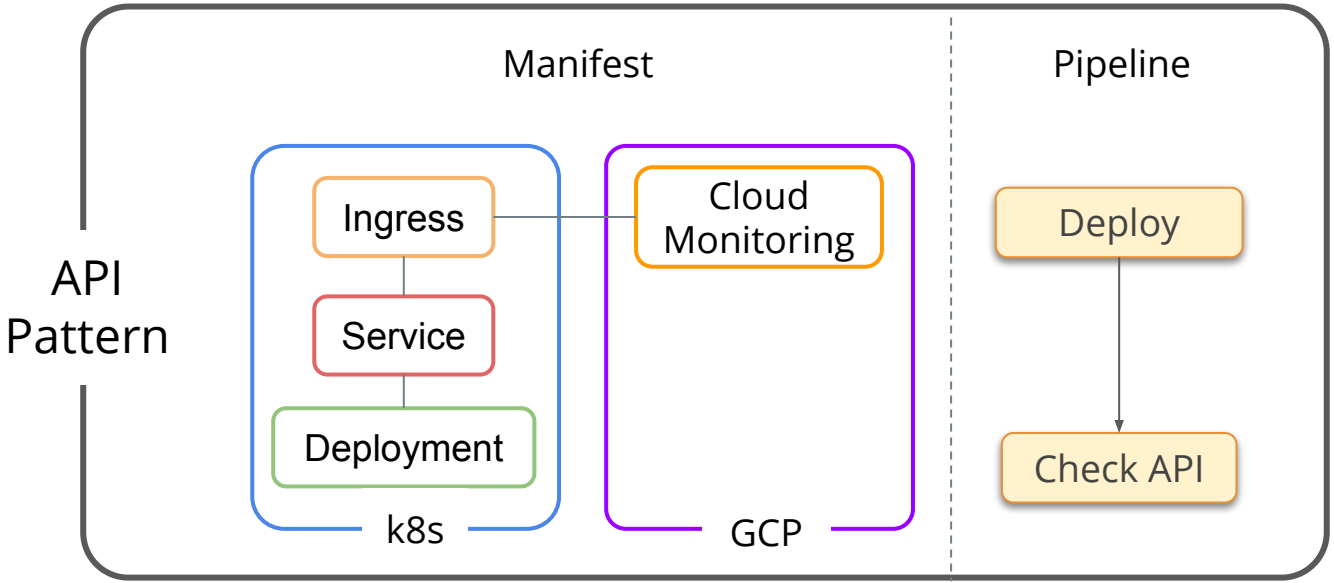

```yaml
kind: Pipeline
apiVersion: tekton.dev/v1beta1
spec:
  tasks:
  - name: build-image
    steps:
    - name: run-buildkit
  - name: deploy
    steps:
    - name: generate-manifest
    - name: kubectl-deploy
    runAfter:
    - build-image
  - name: check-api
    steps:
    - name: check-uptimecheck
    runAfter:
    - deploy
```

Compose pipeline tasks




```yaml
kind: Pipeline
apiVersion: tekton.dev/v1beta1
spec:
  tasks:
  - name: build-image
    steps:
    - name: run-buildkit
  - name: backup-db
    steps:
    - name: export-to-gcs
  - name: deploy
    steps:
    - name: generate-manifest
    - name: kubectl-deploy
    runAfter:
    - build-image
    - backup-db
  - name: check-api
    steps: ...
```

Put together for a API Pattern



API Pattern Implementation Example

The ***domainName*** used for the resource configuration is consistent with the task to check if the API is active.

Design Pattern as Code helps us keep **consistency between configuration and pipeline**.

- Kind of a module per scenario, very basic software approach

```
parameters: domainName: string

resources: {
  kubernetes: ...
  gcp: uptimeCheckConfig: {
      period: "300s"
      monitoredResource: {
        type: "uptime_url"
        labels: host: parameters.domainName
      }
    }
  }
}

tasks: test: check-api: {
  steps: [{
    command: ["curl", parameters.domainName]
  }]
}
```

Demo

Deploy api server with design pattern

Scenario: Launch a microservice by its tech lead

1. Choose Design Pattern
2. Compile it and get a Tekton Pipeline yaml
3. Install Tekton resources
4. Run the Pipeline
5. Show applied results

Update Design Pattern to **expose api endpoint and add monitoring** function.

Capability - Security & Compliance

Create a Design Pattern to add your security checkpoint, like OSS static analysis of container vulnerabilities, to a pipeline. This **keeps consistency on policy enforcement for your application vulnerabilities scanning**. Also empower your dedicated security team by letting them focus on pattern implementation.

clair

trivy

anchore

Source of photo: https://landscape.cncf.io/

Capability - Observability & Analysis

Enable observability and analysis as well with Design Patterns. Install and maintain metrics agents to establish well designed feedback loop. Also, can add a custom metrics exporter base on application requirements.

Prometheus

sysdig

Source of photo: https://landscape.cncf.io/

Capability - Multi Platform

You can leverage any public Cloud specific features by just swapping Design Pattern. Focus in each Design Pattern to provide well-architected best practices specific to each Cloud provider.

Google Cloud

aws

Microsoft Azure

Source of photo: https://landscape.cncf.io/

One step further for Design Pattern as Code

Design Pattern Value Proposition

Design Pattern as Code is a new interface for maximizing product Value Stream.

- To scale with software approach.
- To design architecture including Pipeline.
- To reuse well-architected patterns and reduce Infrastructure management cost per project.

Configuration as Data

Design Pattern as Code aligns with the idea of Configuration as Data, but covers Pipeline as well.

- JSONNET
- Kustermize
- kpt
- Tanka
- and more...

State management

Design Pattern doesn't address state management.

- Additional layer to maintain infrastructure state is required.

Design Pattern as Code

State Management

Terraform

pulumi

Crossplane

kubernetes

Google Cloud

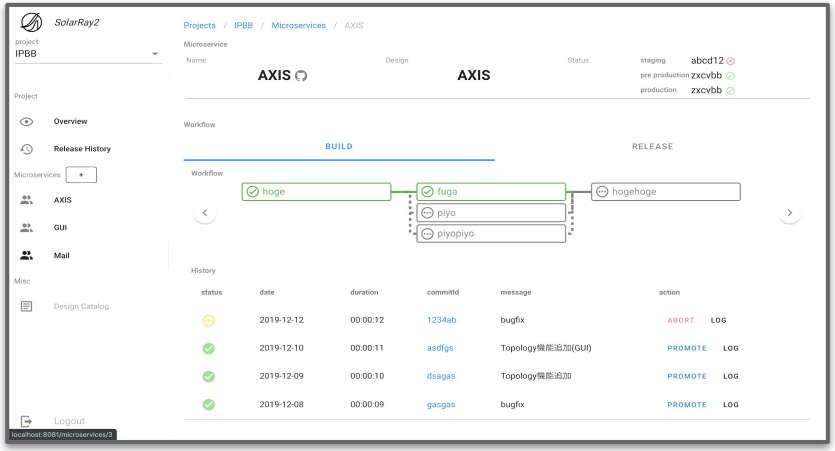
Azure

aws

Our in-house platform

A DevOps platform on top of Design Pattern as Code.

- Bakes in state management with Pulumi.
- Automate all operations for Design Pattern as Code.
  - To let developer easily subscribe patterns and truly focus on application development.



Wrap up

Design Pattern as Code

- **Reusable** and **composable** pattern of Cloud Native architecture.
- Includes all infrastructure providers' **Manifest** and **Pipeline**.
- Addresses **consistency and reusability** for the infrastructure Manifest and its delivery Pipeline.

Thank you

Reach out to us at @JunMakishi or @TAR_O_RIN

Demo code:
https://github.com/j-maxi/designpattern-as-code
https://github.com/j-maxi/simple-app