
KubeCon
CloudNativeCon
Europe 2023
TiKV

AUTODESK

Building a Platform Engineering Fabric w/ the Kube API at Autodesk

Jesse Sanford
Sr Principal Software Engineer | @jessesanford
Greg Haynes
Software Architect

Presenters

Jesse Sanford

Sr Principal Software Engineer

Focus on Platform Engineering and Software Supply Chain Security

Sailor, Backcountry skier and continuously delivering father

Greg Haynes

Software Architect / OSS Lead

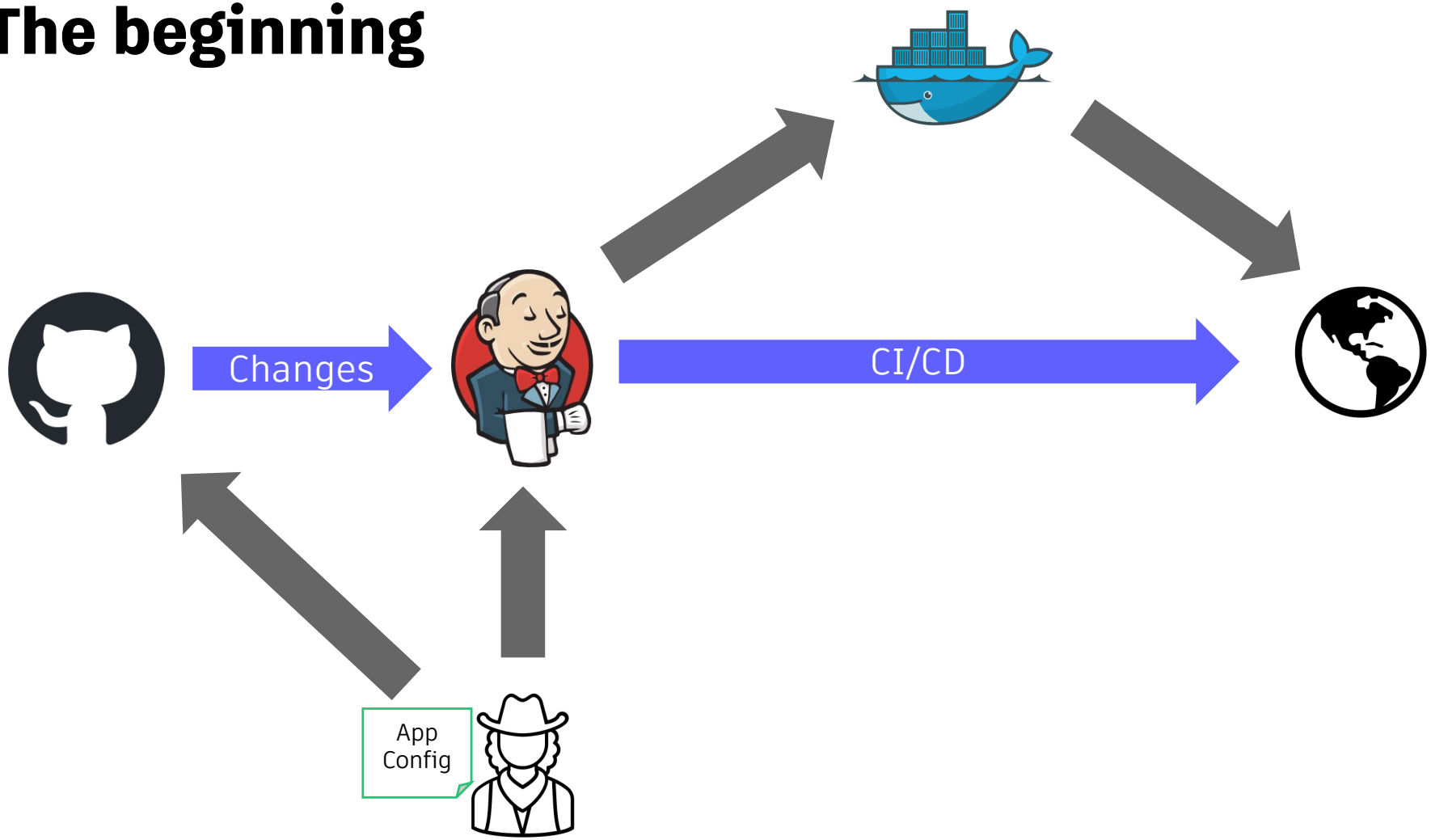
Focused on Internal Developer Platform technology

10+ Years of OSS Cloud Tech. Development, including Knative and OpenStack.

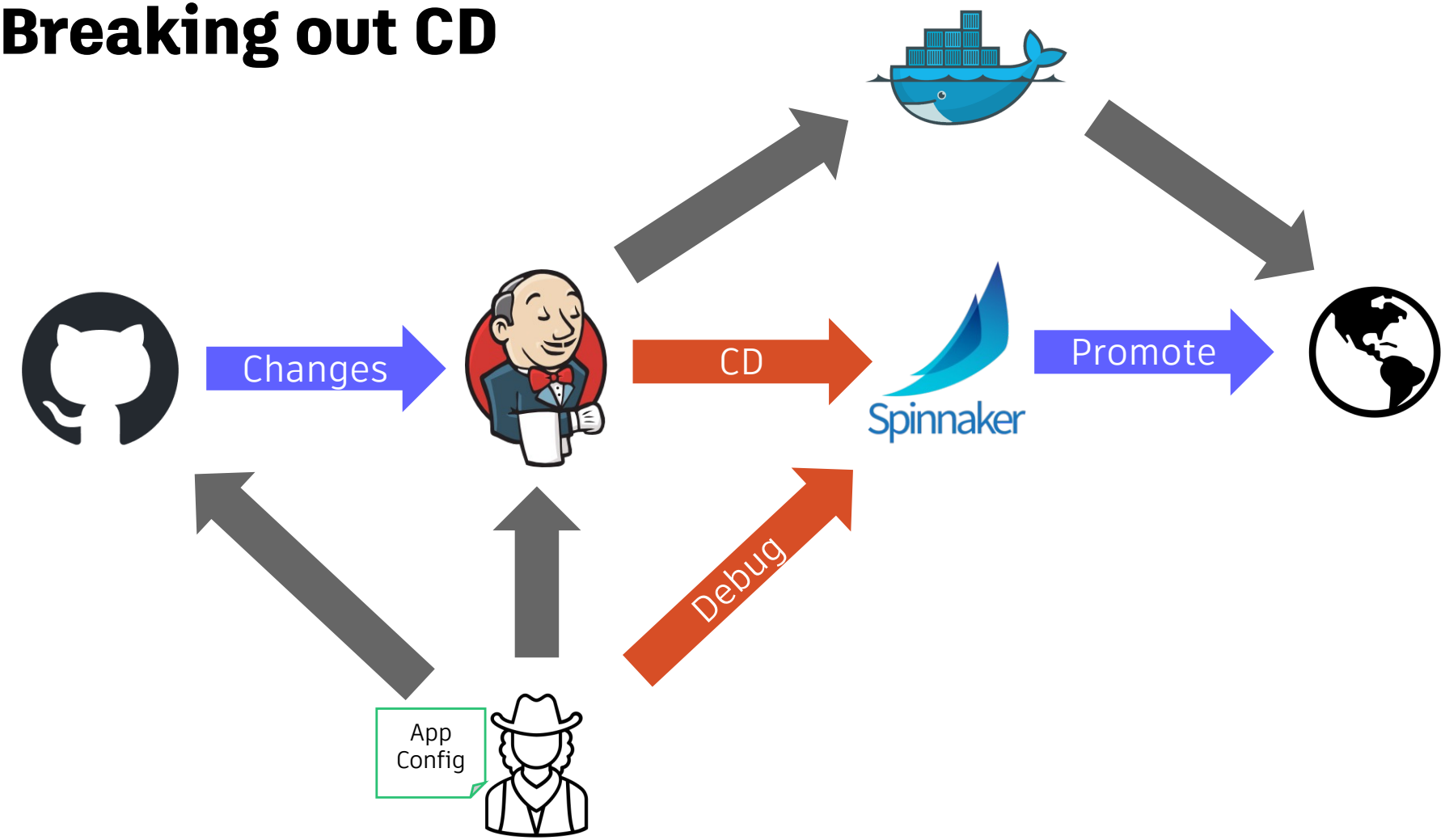
Agenda

1. Our Initial Journey
2. K8s-API North Star
3. Choices, Choices
4. Challenges and Solutions
5. Lessons Learned

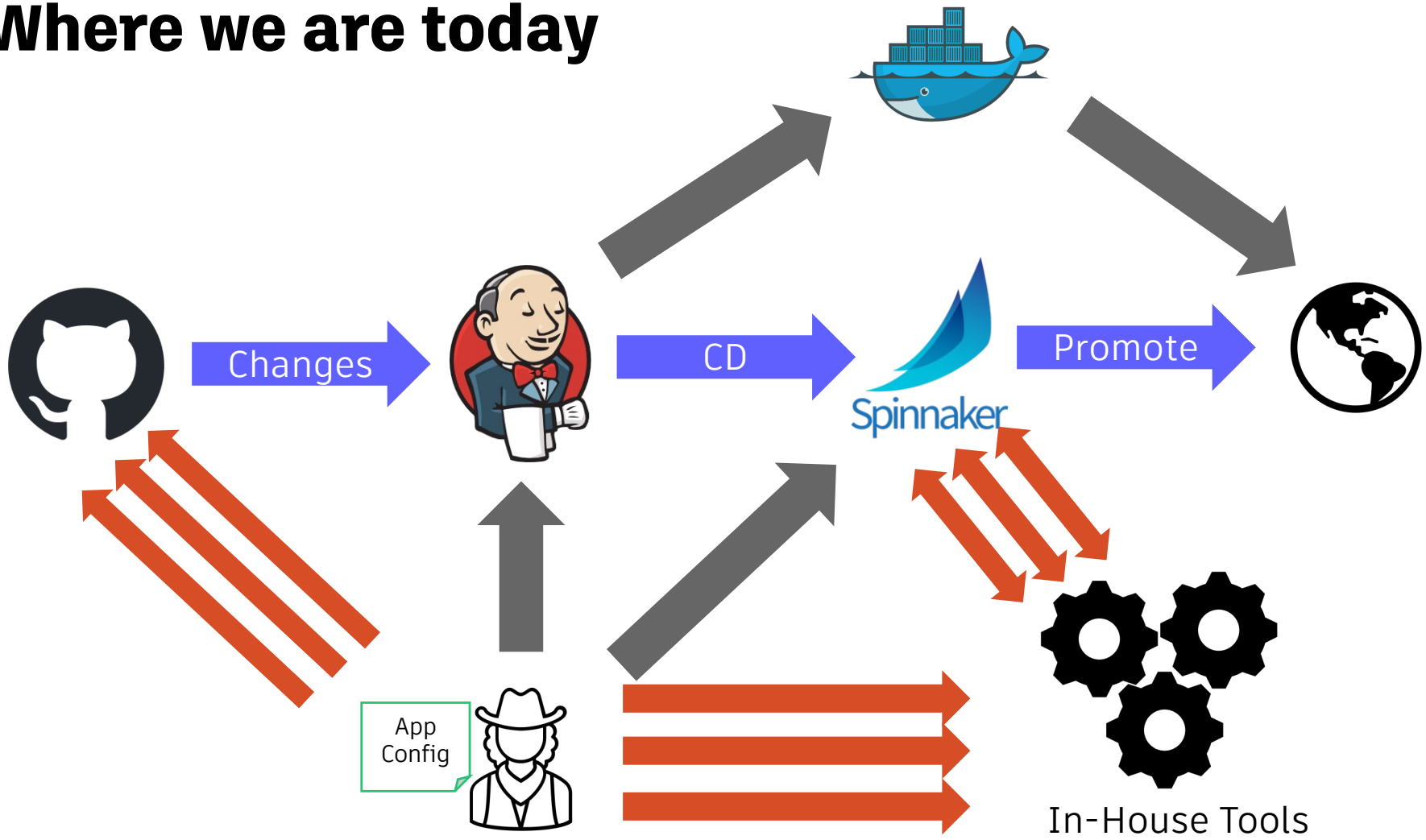
The beginning



Breaking out CD

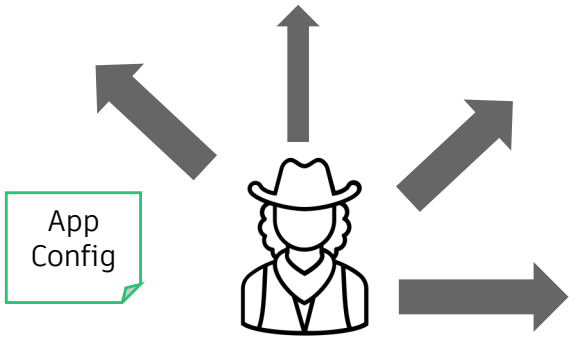


Where we are today



Pain Points

- N different user interfaces
  - Remember all those arrows!?
  - App config is yet another interface



- Integration Nightmare
  - Struggles maintaining velocity
  - Try implementing a "developer portal"

Backstage

- What is the ownership model?
  - Devs owning "rollout" is a scalability issue

Vision

K8s-API platform fabric

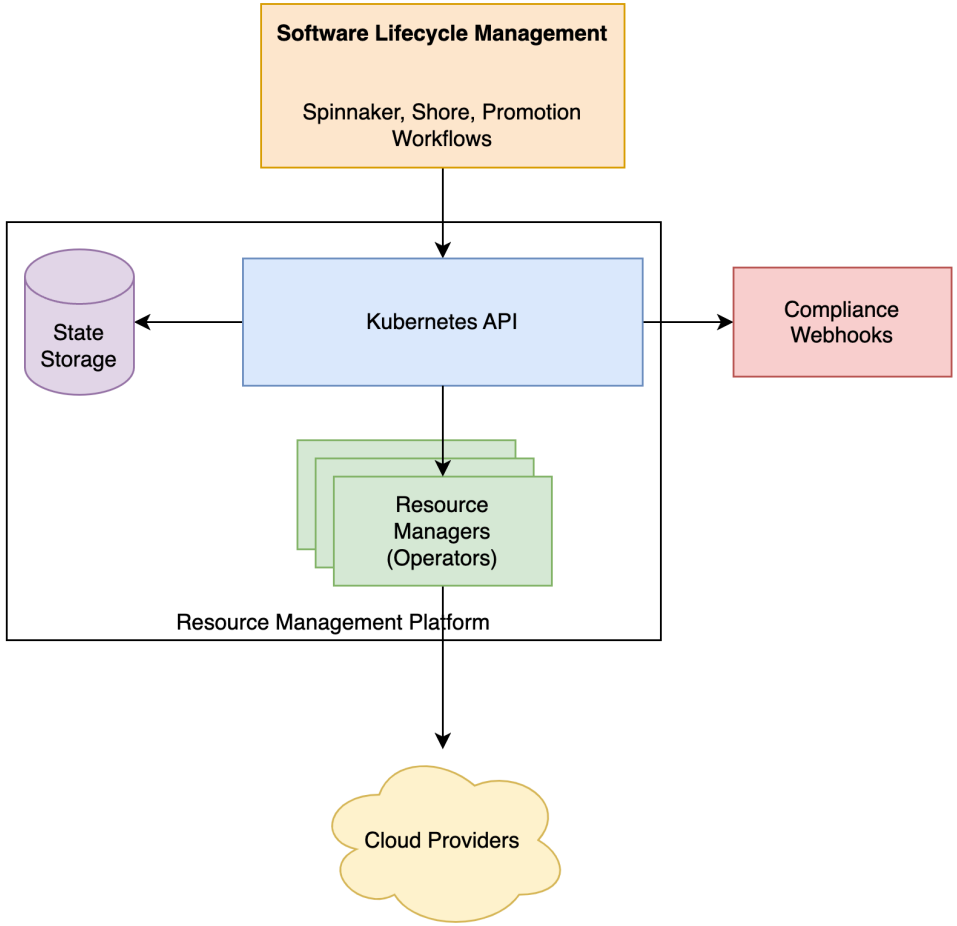
Capabilities as Operators:

- Continuous Reconciliation
- Patterns for integration
- Clear Separation of Concerns
  - Hermetic Abstractions
- Enables Inner Source

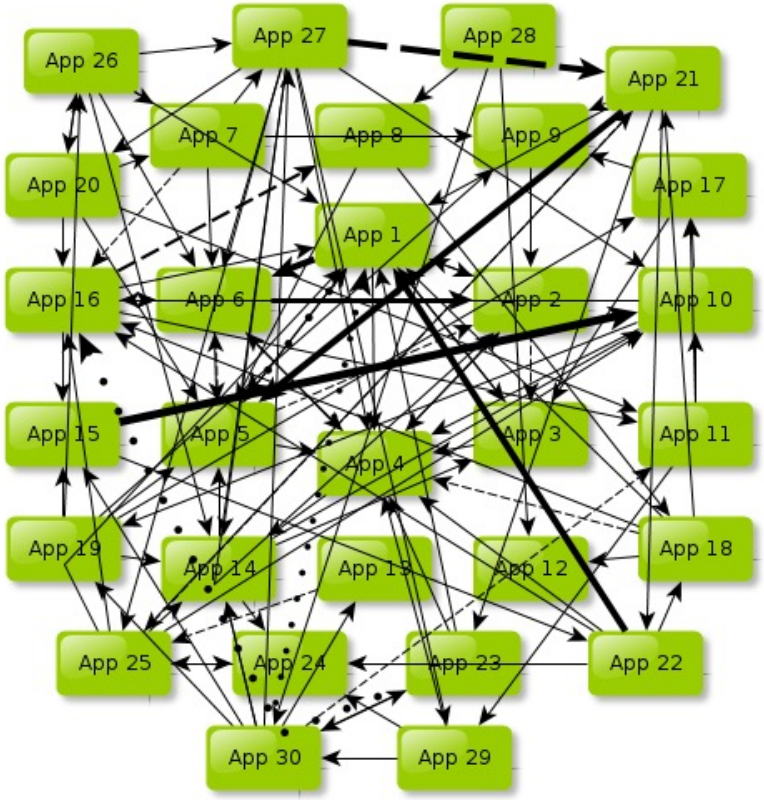
Kube-API and KRM:

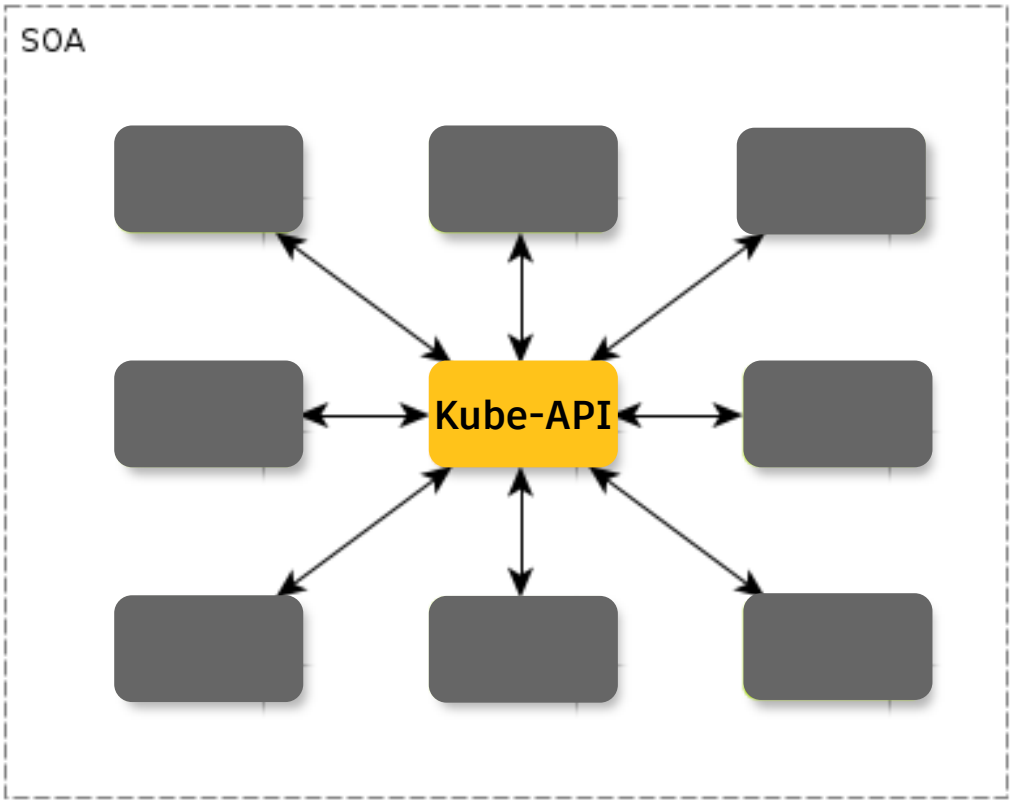
- Strong API semantics
  - Validation
  - Versioning and Conversions
- RBAC and Name Spacing
- Unified Governance Plane
- Free CMDB / Inventory DB

K8s-API platform fabric

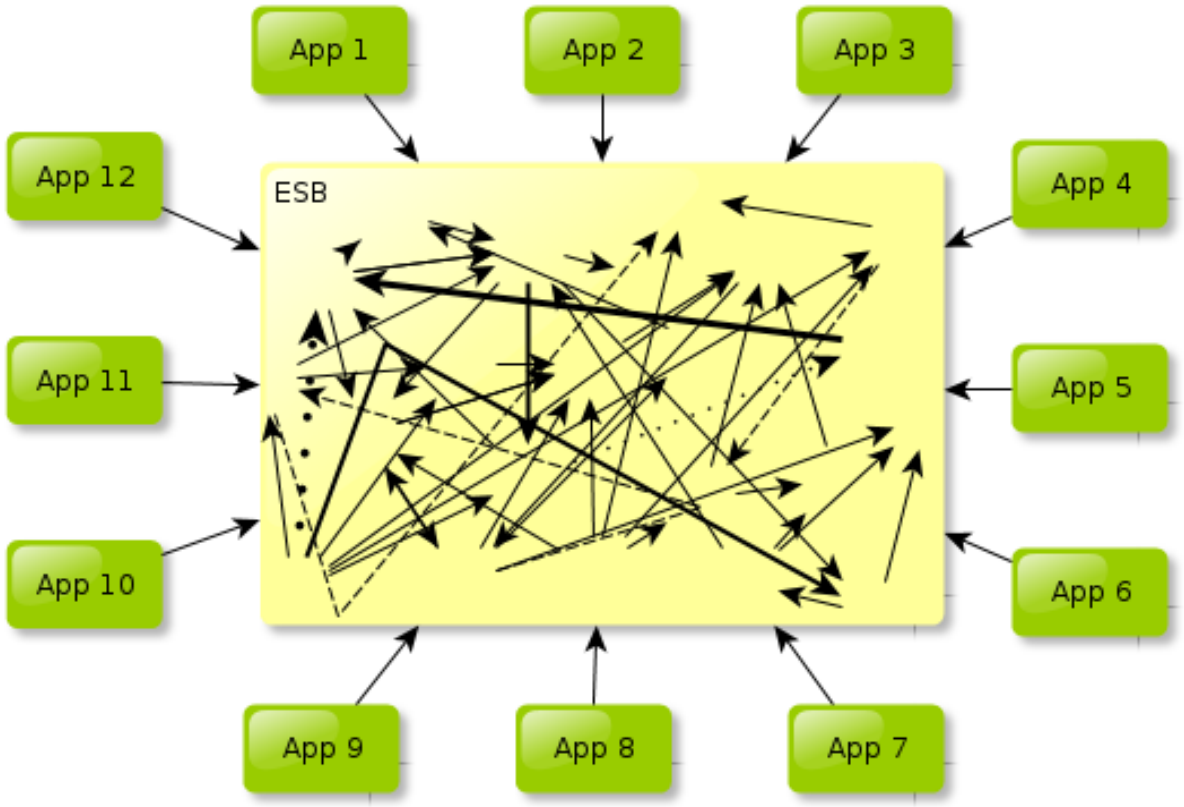


K8s-API platform fabric

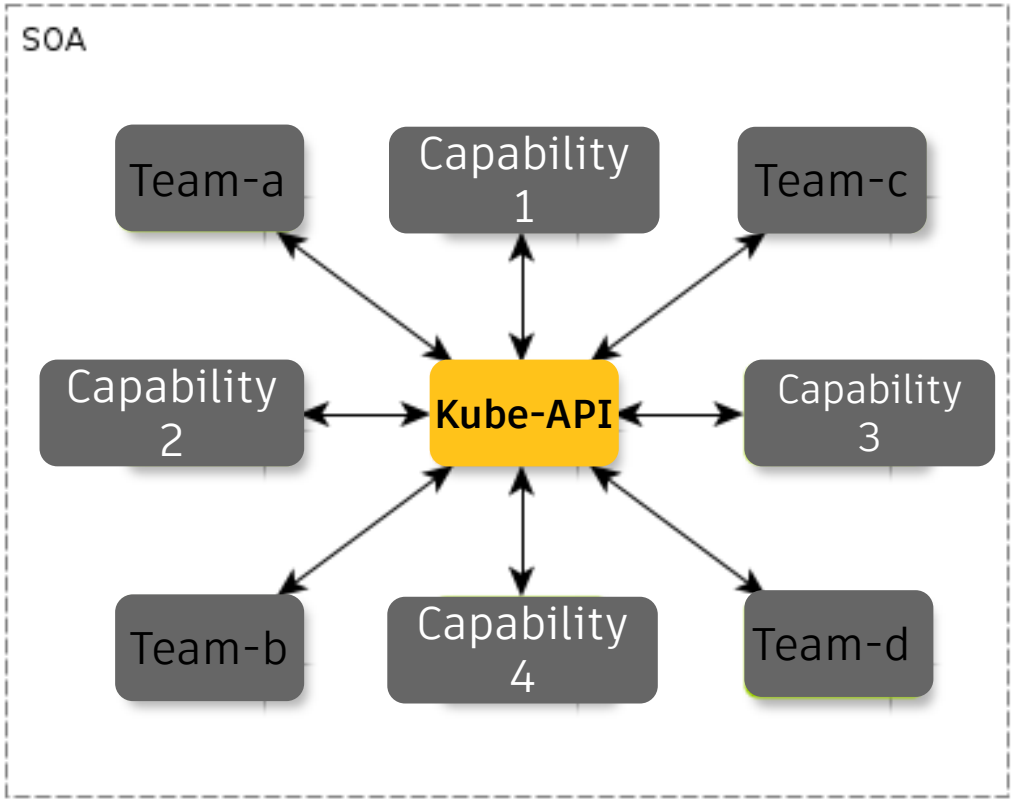




Kube-API ESB

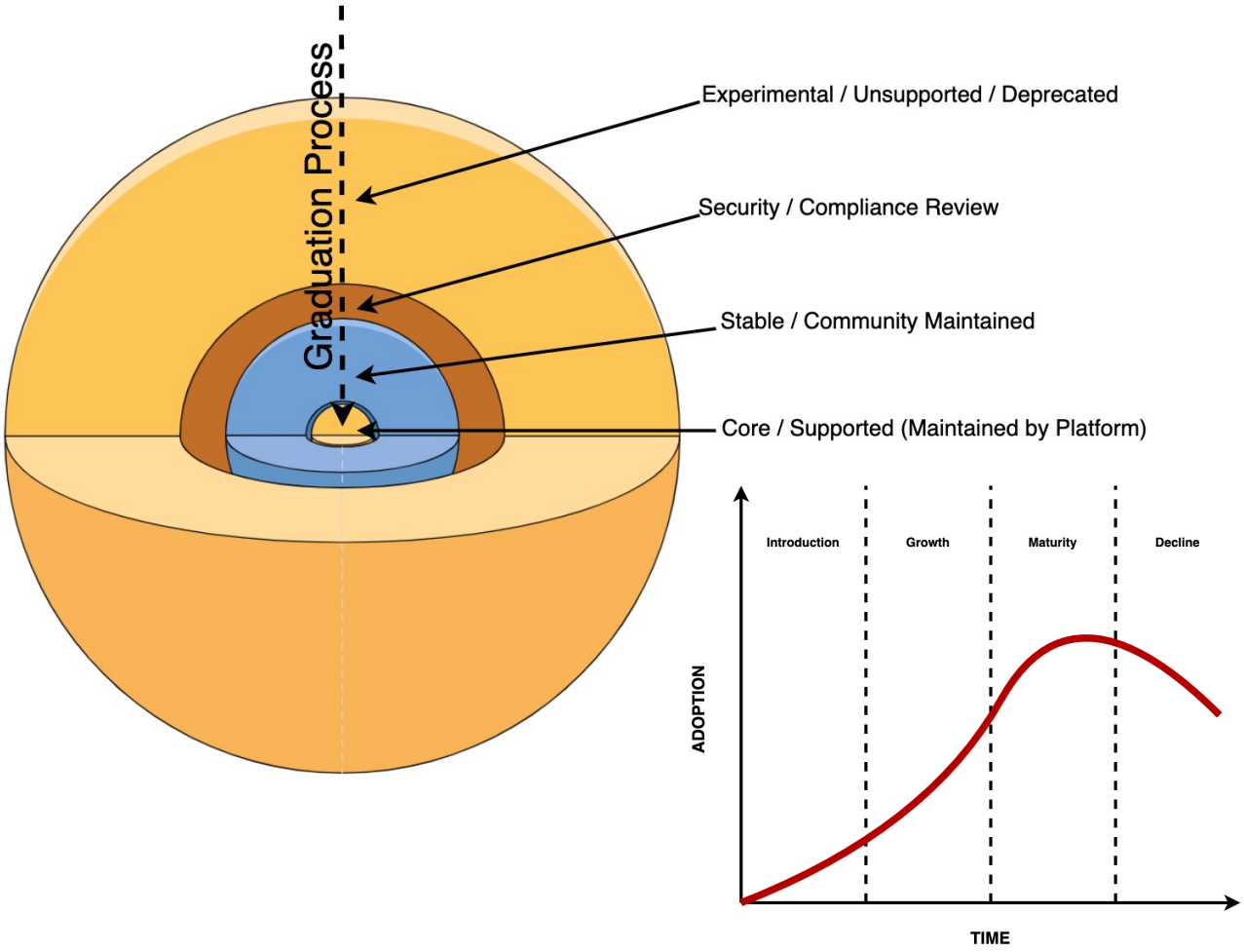

App 1
App 2
App 3
App 12
ESB
App 4
App 11
App 5
App 10
App 6
App 9
App 8
App 7

K8s-API platform fabric



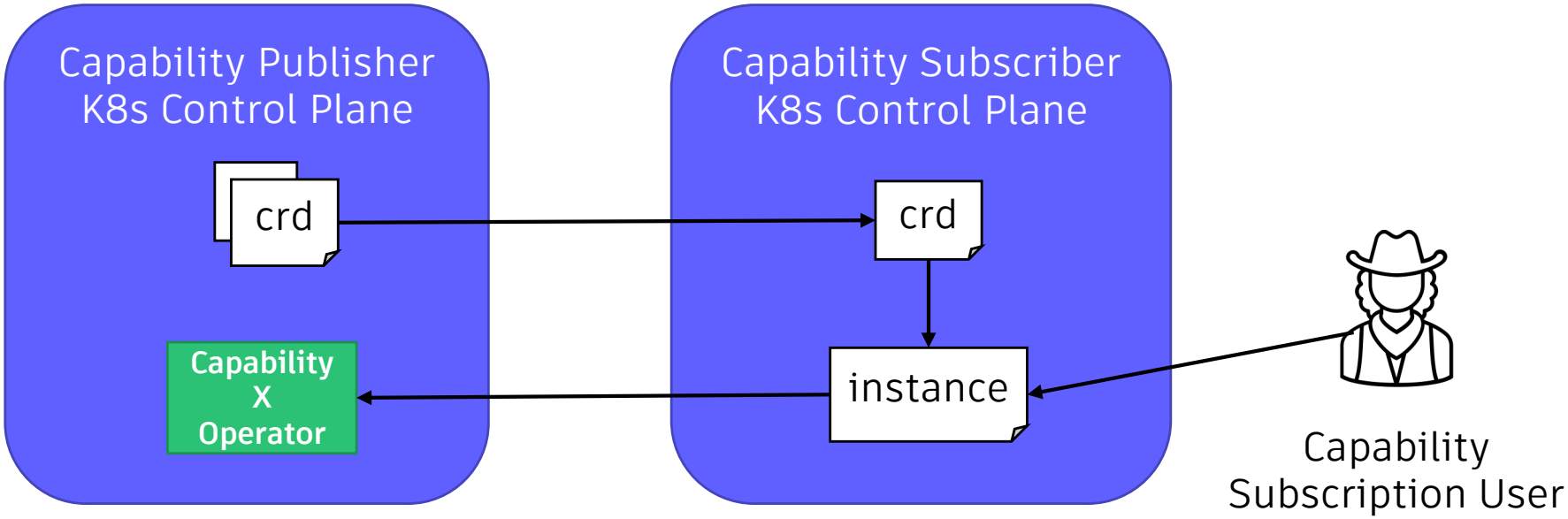
Governance and maintenance

- Decoupled capability SDLCs
- KRM versioning semantics
- Subscription model
- Security and compliance programs

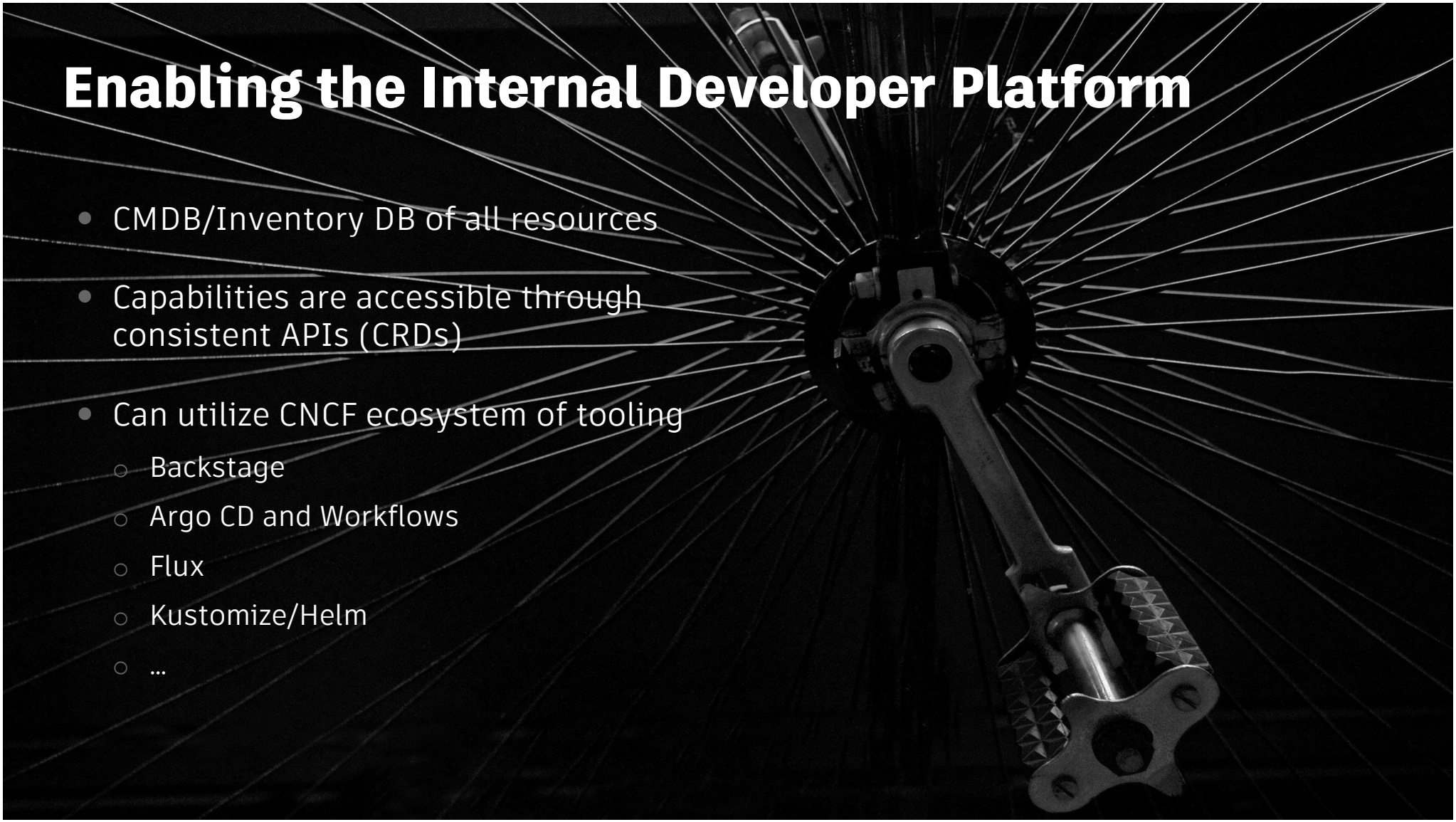


K8s-API platform fabric

Network of Capabilities?



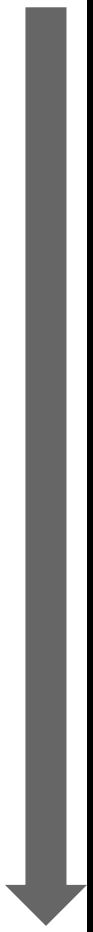
Enabling the Internal Developer Platform



- CMDB/Inventory DB of all resources
- Capabilities are accessible through consistent APIs (CRDs)
- Can utilize CNCF ecosystem of tooling
  - Backstage
  - Argo CD and Workflows
  - Flux
  - Kustomize/Helm
  - ...

Top Down

- Completely Enabled Vision
  - Fully Declarative App Delivery
- Complex functionality must be dealt with early:
  - Routing
  - Progressive Deployments
  - Stateful services and migrations

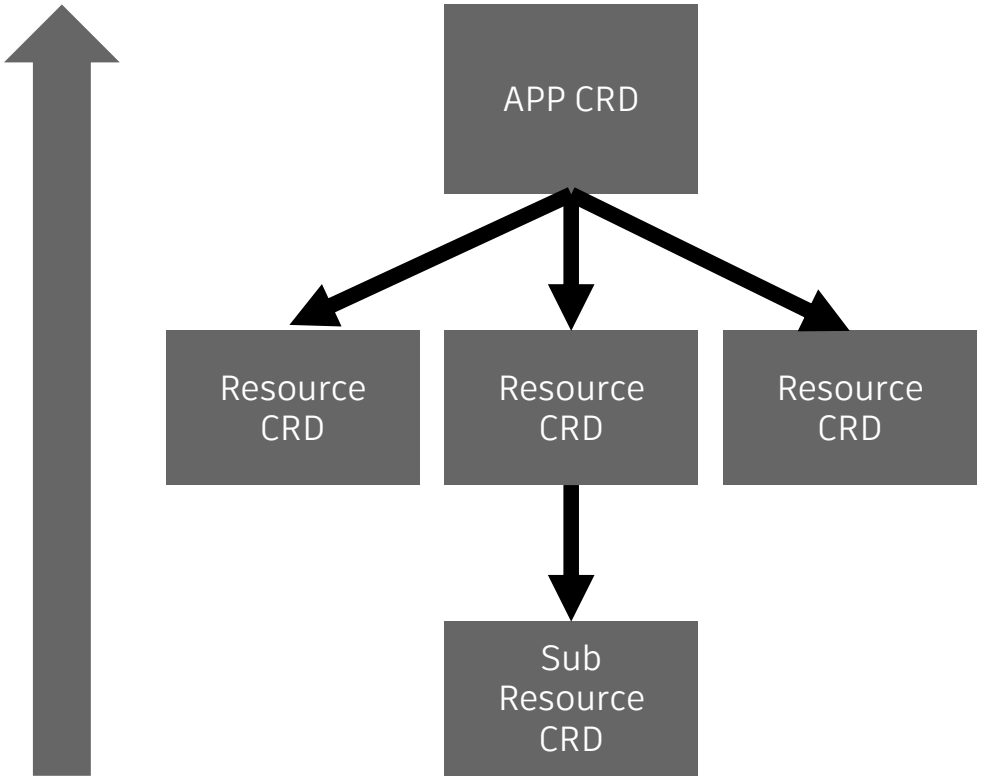


Bottom Up

- Small Encapsulated Resources
  - Easy to understand
  - Can be “Shimmed” into existing solution
- Doesn’t fully validate tooling
- Might not be powerful enough

Attacking the Problem: Bottom Up

- Smaller, atomic, “managed resources”
- Easier to learn and ramp up engineers
- Can make common SDLC decisions early
  - Repo structure
  - Developer environments
  - CI/CD automation
  - Testing

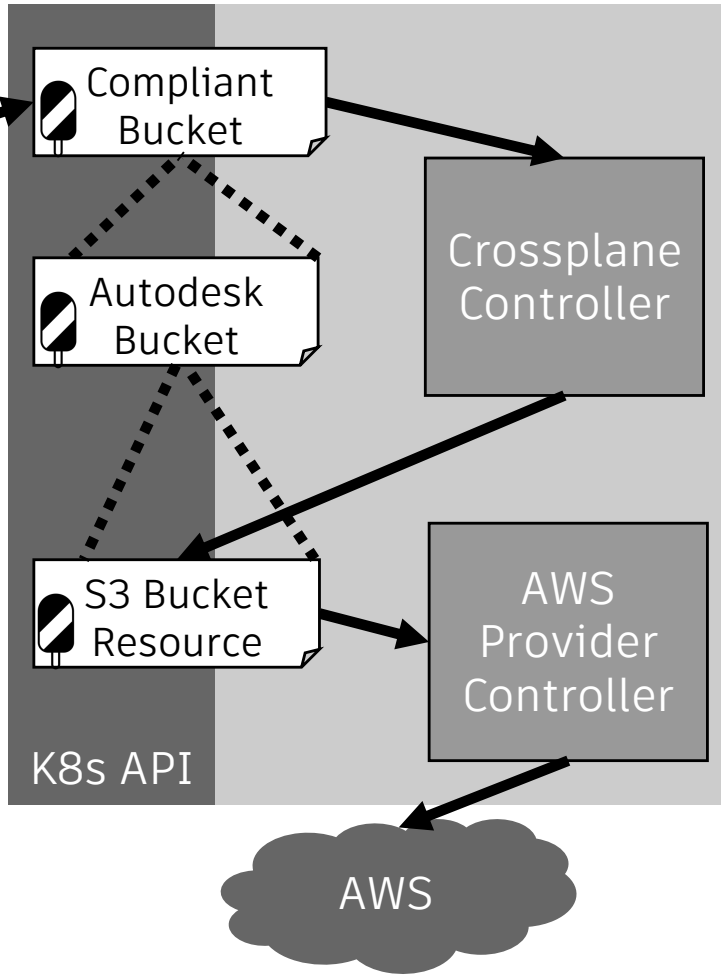


Attacking the Problem: Bottom Up

Crossplane

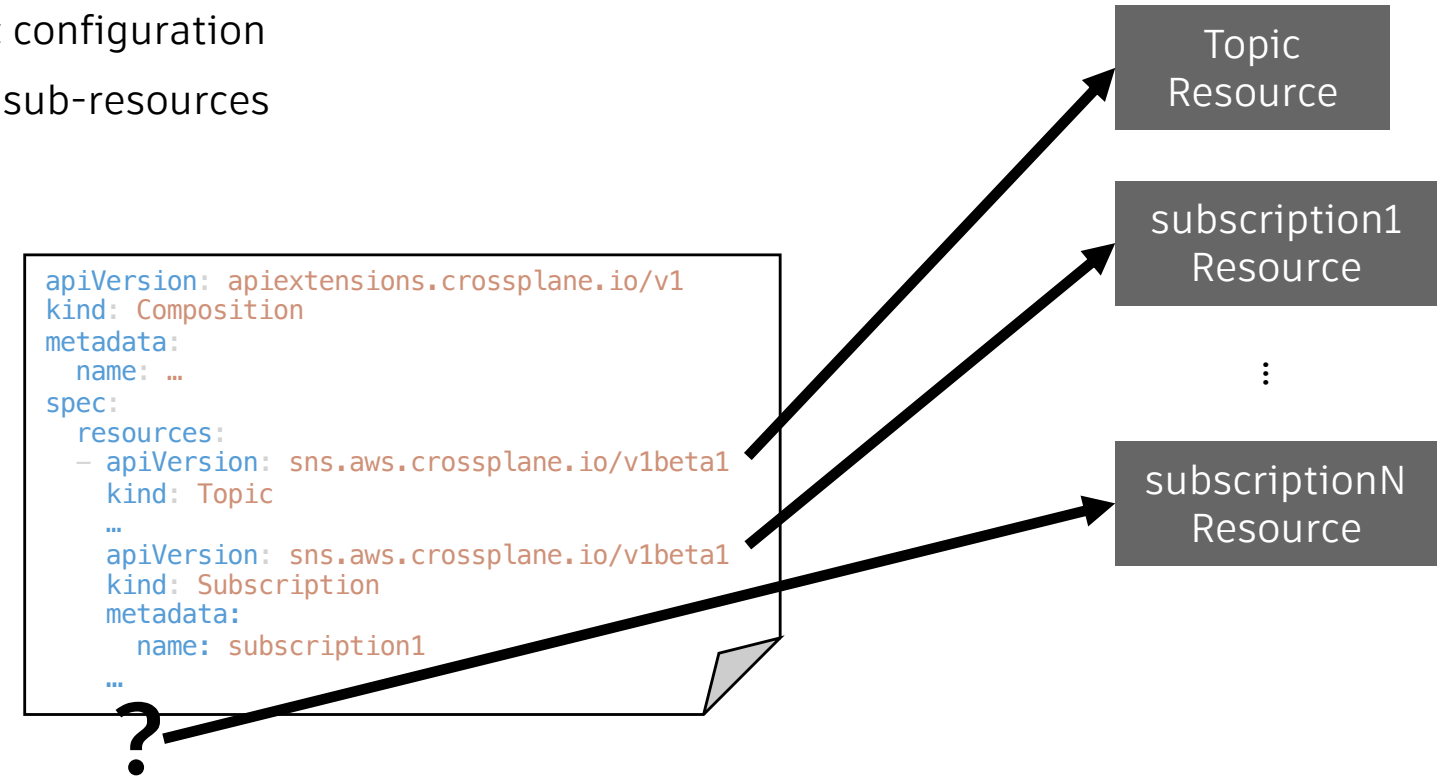
- Has library of providers
- Higher order types
  - Sane Defaults
  - Security and Compliance baked in
- Registers native Kubernetes types
  - Replace w/ own operator (break glass)
  - K8s ecosystem tooling

```
apiVersion: bucket.demo.com/v1
kind: CompliantBucket
metadata:
  name: test-topic
spec:
  bucketName: Test
```



Complexity problem

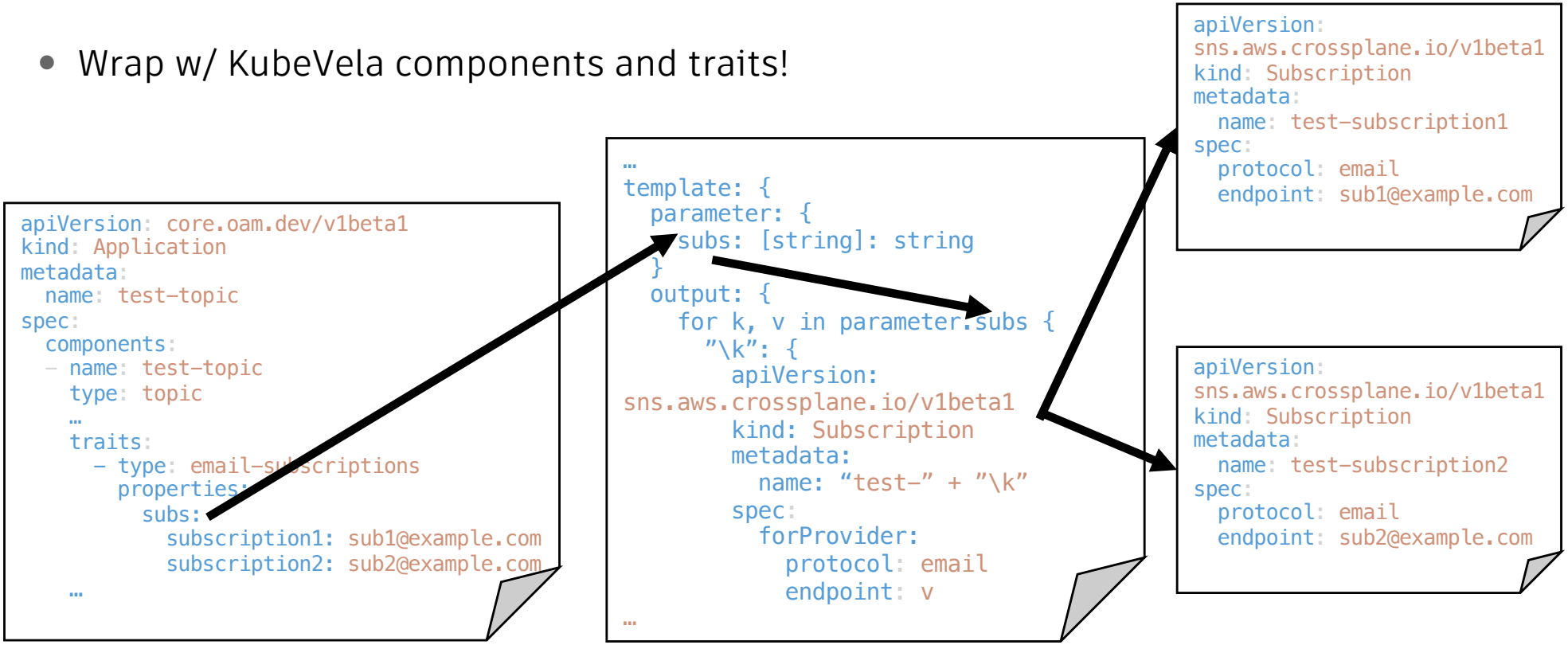
- How can we handle complex resource usage?
  - Optional or dynamic configuration
  - Varying numbers of sub-resources
- SNS Example:



Something in between?

KubeVela

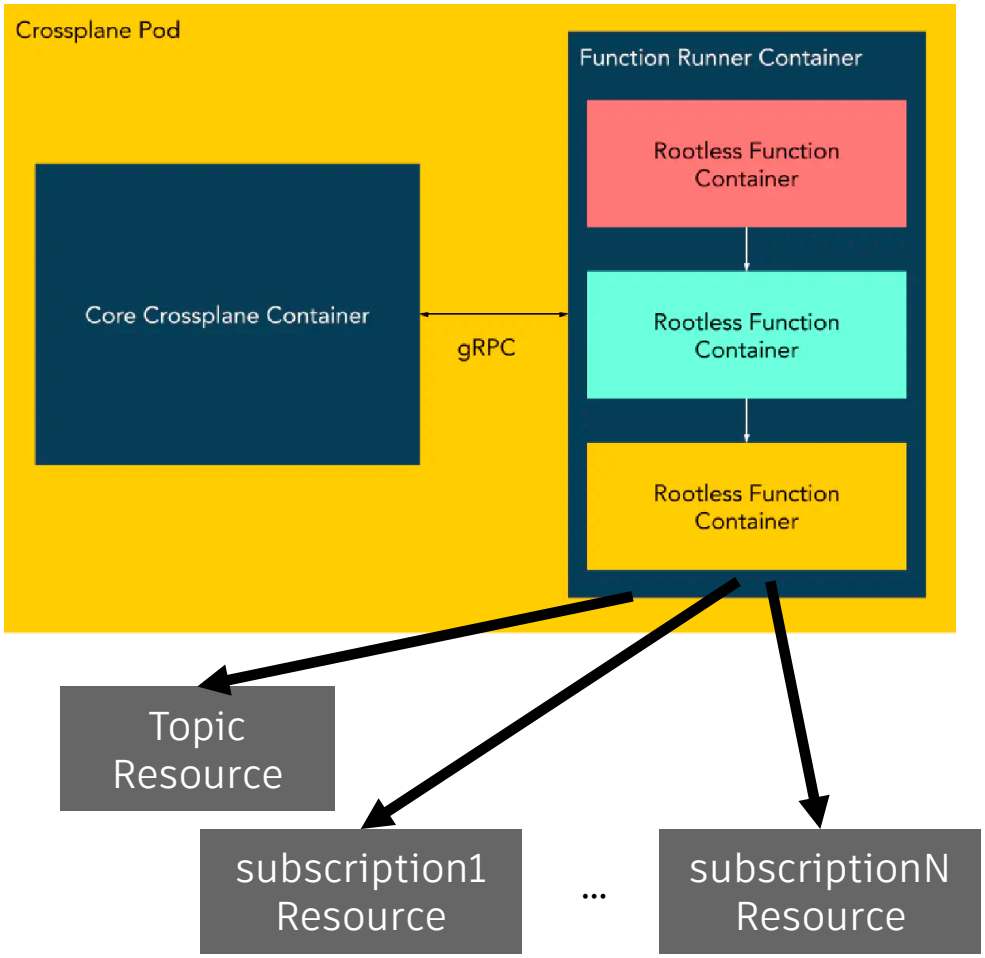
- Still use Crossplane Compositions and Resources
- Wrap w/ KubeVela components and traits!



Future? Composition Functions

- Bleeding edge
- Similar to KPT functions
- Hides a lot from vs pure declarative

```yaml
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: …
spec:
  resources:
  - apiVersion: sns.aws.crossplane.io/v1beta1
    kind: Topic
    …
    patches:
     …
   functions:
   - name: add-subs
     type: Container
     container: myreg/xp-fns/add-subs:v0.1.0
```



Solutions!

- On our journey, we encountered a lot of areas without established best practices.
- Although specific technology is mentioned, these are all general architecture / design problems.
- These *should* apply regardless of the technology used on top of Kube-API.

Themes:

- Kube-API and controllers are mature, but their use in this space is not.
- These are areas we'd love to see the wider community form standards!

Owning our API

AKA: Don’t leak implementation

```yaml
apiVersion: sns.mycorp.com/v1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
  cloudAccount: 1234
```

IDP API

```yaml
apiVersion: sns.aws.crossplane.io/v1beta1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
  providerConfigRef:
    name: cloudprovider-12345
  region: us-west-2
```

Implementation

Owning our API

AKA: Don’t leak implementation

```yaml
apiVersion: sns.mycorp.com/v1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
  cloudAccount: 1234
```

IDP API

```yaml
apiVersion: sns.aws.crossplane.io/v1beta1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
  providerConfigRef:
    name: cloudprovider-12345
  region: us-west-2
```

Implementation

Owning our API

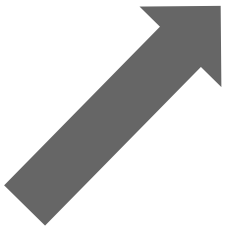
AKA: Don’t leak implementation

```yaml
apiVersion: sns.mycorp.com/v1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
  cloudAccount: 1234
```

IDP API

```yaml
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: …
spec:
  resources:
  - apiVersion: sns.aws.crossplane.io/v1beta1
    kind: Topic
    …
    patches:
     …
```

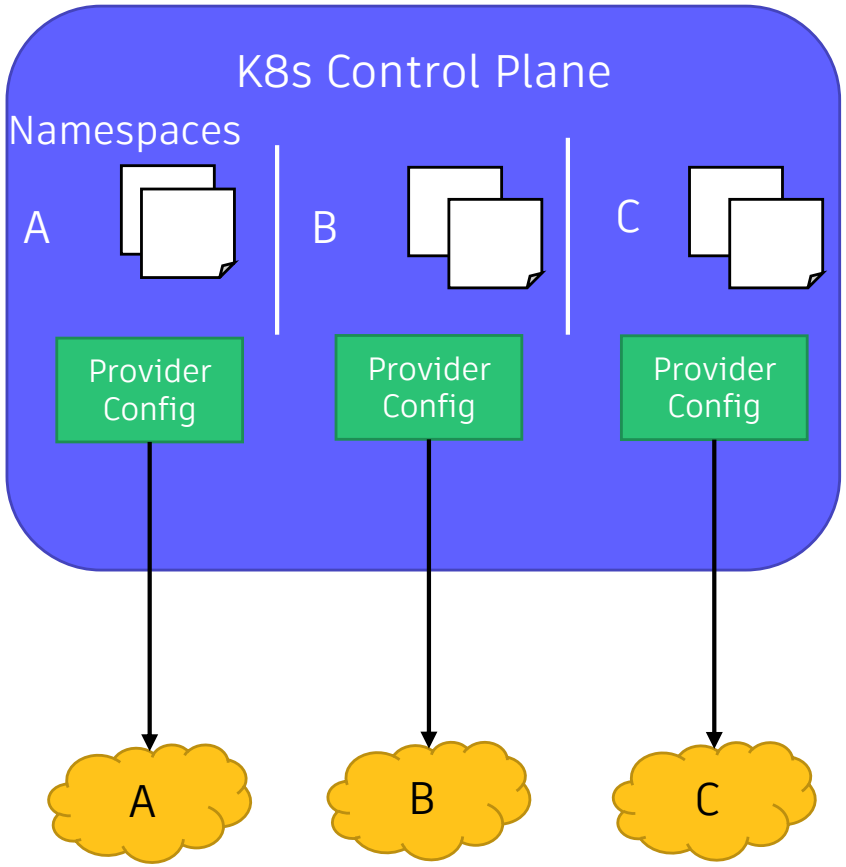
Indirection (Composition)



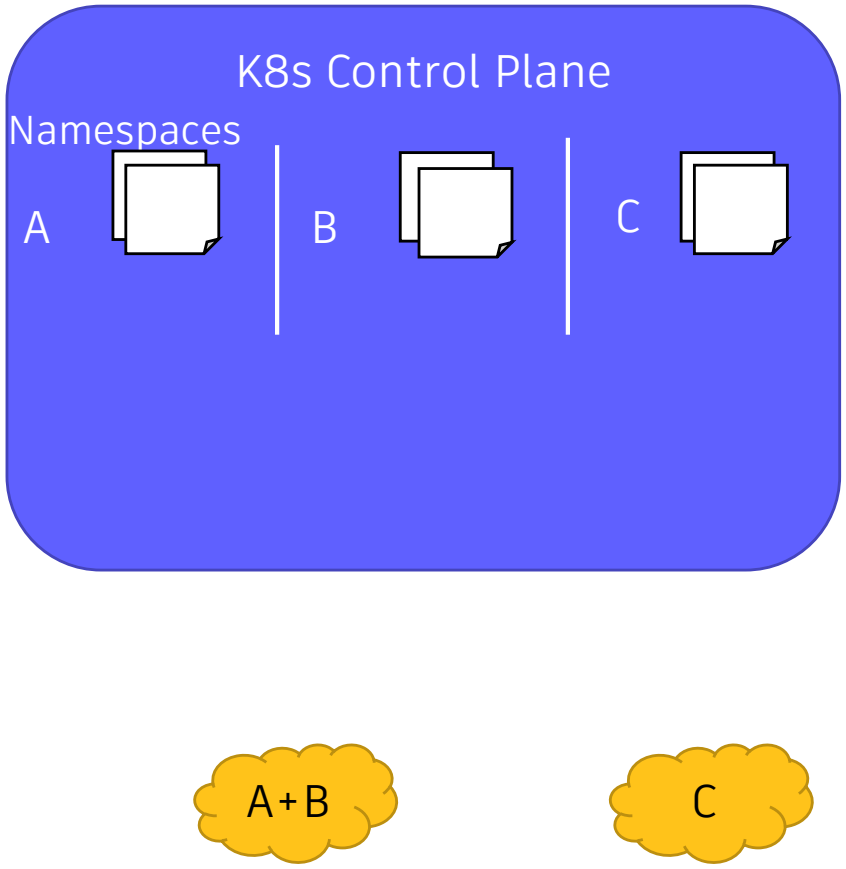
```yaml
apiVersion: sns.aws.crossplane.io/v1beta1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
  region: us-west-2
  providerConfigRef:
    name: cloudprovider-12345
```

Implementation

Accounts / Tenancy



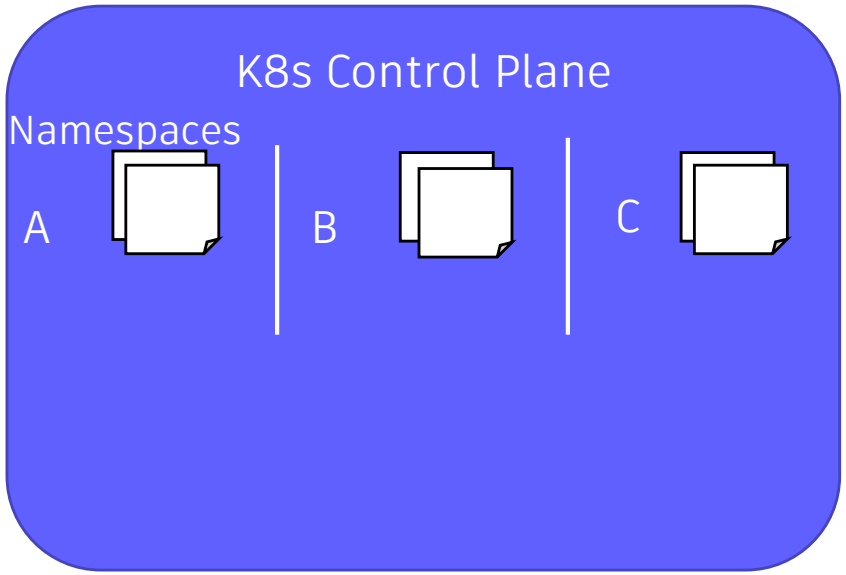
Accounts / Tenancy



Accounts / Tenancy

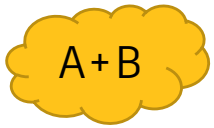
Dev Platform Tenancy

Namespace Design Restriction



Cloud Provider Tenancy

Provider-Based Design Restriction



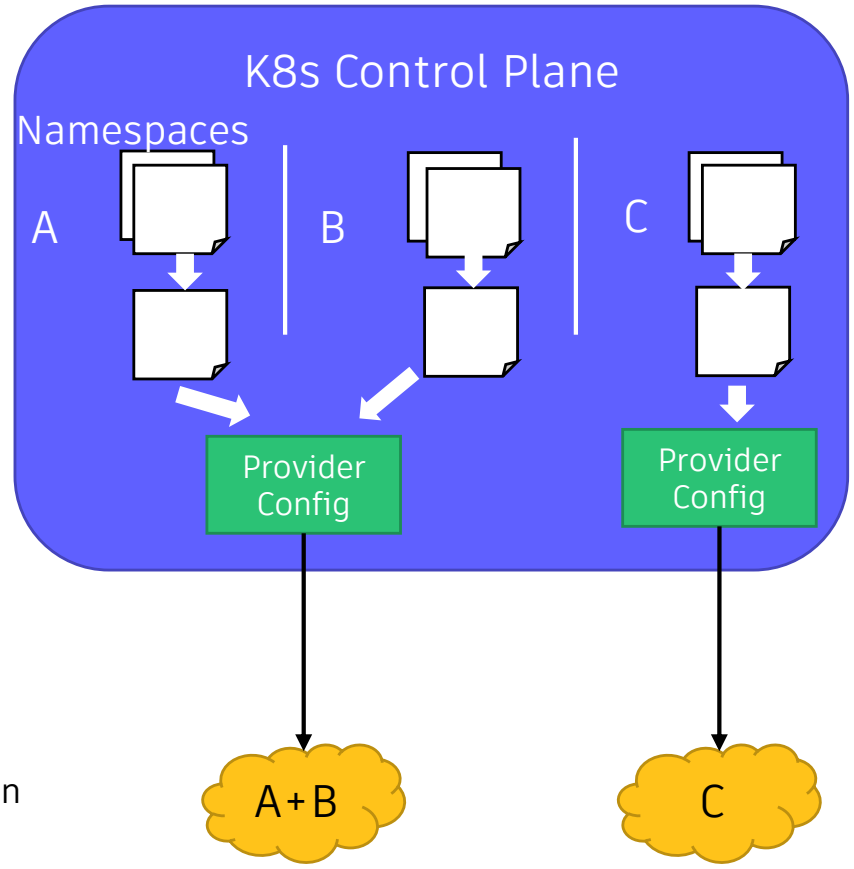
Accounts / Tenancy

Dev Platform Tenancy

Namespace Design Restriction

Cloud Provider Tenancy

Provider-Based Design Restriction



Compositions / Implementation

Resource Referencing

```yaml
apiVersion: sns.mycorp.com/v1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
```

```yaml
apiVersion: sns.mycorp.com/v1
kind: Subscription
metadata:
  name: test-subscription
spec:
  topic:
    name: test-topic
  protocol: email
  endpoint: null@example.com
```

Resource Referencing

```yaml
apiVersion: sns.mycorp.com/v1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
```

```yaml
apiVersion: sns.mycorp.com/v1
kind: Subscription
metadata:
  name: test-subscription
spec:
  topic:
    name: test-topic
  protocol: email
  endpoint: null@example.com
```

Composition / Indirection

```yaml
apiVersion: sns.aws.crossplane.io/v1beta1
kind: Topic
metadata:
  name: test-topic-12345
spec:
  displayName: Test Topic
  region: us-west-2
```

```yaml
apiVersion: sns.aws.crossplane.io/v1beta1
kind: Subscription
metadata:
  name: test-Subscription-12345
spec:
  displayName: Test Topic
  region: us-west-2
  forProvider:
    topicArnRef: …
```

Resource Referencing

```yaml
apiVersion: sns.mycorp.com/v1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
```

```yaml
apiVersion: sns.aws.crossplane.io/v1beta1
kind: Topic
metadata:
  name: test-topic-12345
spec:
  displayName: Test Topic
  region: us-west-2
```

```yaml
apiVersion: sns.mycorp.com/v1
kind: Subscription
metadata:
  name: test-subscription
spec:
  topic:
    name: test-topic
  protocol: email
  endpoint: null@example.com
```

How can we Reference ???

```yaml
apiVersion: sns.aws.crossplane.io/v1beta1
kind: Subscription
metadata:
  name: test-Subscription-12345
spec:
  displayName: Test Topic
  region: us-west-2
  forProvider:
    topicArnRef: …
```

Resource Referencing

```yaml
apiVersion: sns.mycorp.com/v1
kind: Topic
metadata:
  name: test-topic
spec:
  displayName: Test Topic
```

```yaml
apiVersion: sns.aws.crossplane.io/v1beta1
kind: Topic
metadata:
  name: test-topic-12345
  crossplane.io/claim-name: test-topic
  crossplane.io/claim-name: test-namespace
spec:
  displayName: Test Topic
  region: us-west-2
```

```yaml
apiVersion: sns.mycorp.com/v1
kind: Subscription
metadata:
  name: test-subscription
spec:
  topic:
    name: test-topic
  protocol: email
  endpoint: null@example.com
```

Selectors!

```yaml
apiVersion: sns.aws.crossplane.io/v1beta1
kind: Subscription
metadata:
  name: test-Subscription-12345
spec:
  displayName: Test Topic
  region: us-west-2
  forProvider:
    topicArnSelector:
      matchLabels:
        crossplane.io/claim-name: test-topic
        crossplane.io/claim-name: test-namespace
```

Takeaways

Maturity has a unique meaning here

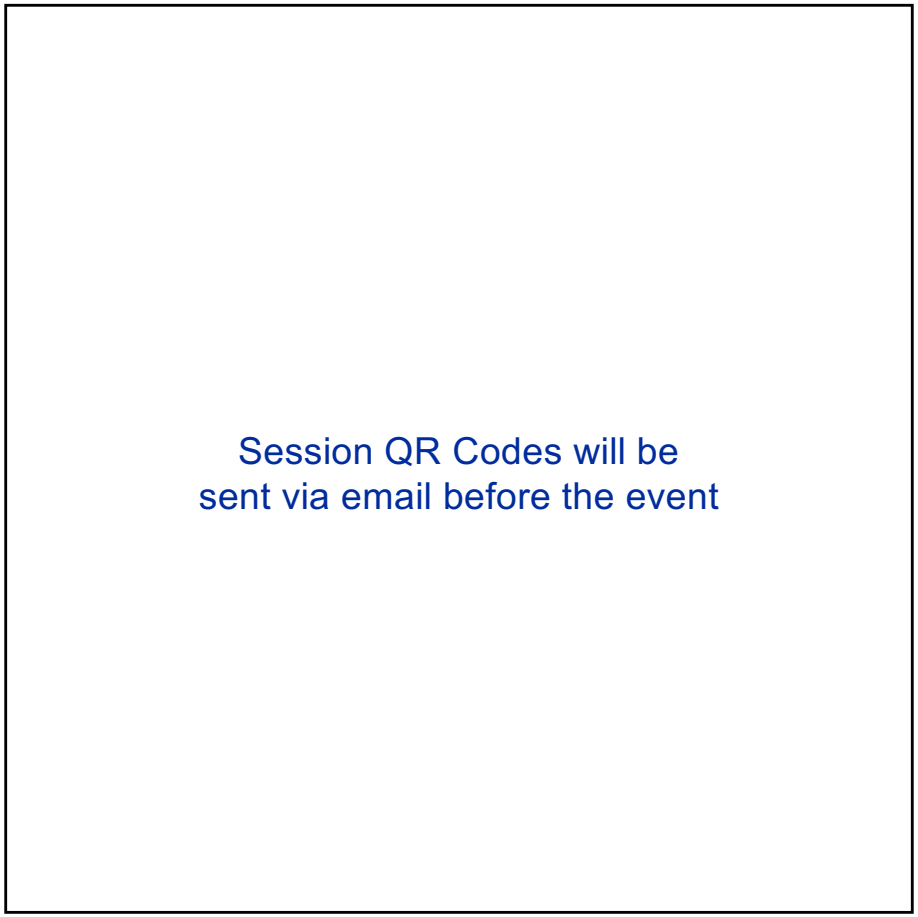
- Kube-API is mature, but its use for a dev platform API is not.
  - Avoid typical new software problems of missing docs, features, or validation for many cases
  - But these problems exist in use-case specific scenarios

Mature	Growth Needed	Examples
Kube-API / Controller Docs	Dev Platform Targeted Docs	• Ownership Models • Isolation • API Design
Kube-API Server	IaC Dev Platform Framework Software	• Crossplane / ACK • Kubevela / OAM

Takeaways

Account for the paradigm shift

- Lots of room for frustration while building familiarity
  - Envision switching programming languages
  - Alignment on value across the org is critical here
- Hard to avoid "lift-and-shift"
  - Copying existing Terraform interfaces = recipe for pain, but this will be the default
  - **API design is often the hard problem, don't shy away from this**.
- Not everyone thrives in ambiguity
  - A small team that can pave a path might be more effective than a large team early on



Please scan the QR Code above
to leave feedback on this session