# AWS Controllers for Kubernetes

The AWS universe of services, now Kubeified.

Jay Pipes, Principal Open Source Engineer
@jaypipes

## A familiar story...

Alice is a web developer and a huge Kubernetes fan.

She's developed a node.js application and is building her application into a Docker image.

The application uses a SQLite database for simple storage.

## Alice goes to deploy on Kubernetes…

```
kubectl apply -f deployment.yml
kubectl apply -f service.yml
```

and probably…

```
kubectl apply -f ingress-nginx.yml
```

## Everything is great. Until…

Ten users try using the site at once.

SQLite falls over.

Alice needs to set up a real database.

## So, Alice sets up a real database…

```
kubectl apply -f postgres-secret.yml
kubectl apply -f postgres-volume-claim.yml
kubectl apply -f postgres-deployment.yml
kubectl apply -f postgres-service.yml
```

## Hmm…

Now Alice is in the RDBMS administration game.

Definitely not what Alice had in mind.

So, what to do?

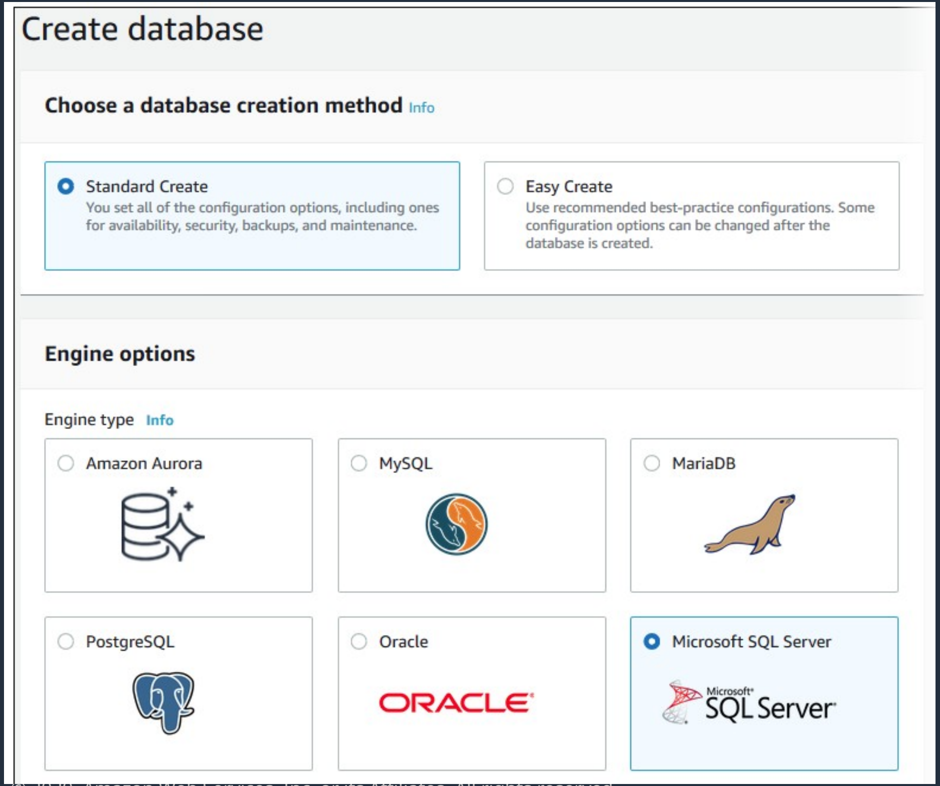
## AWS to the rescue!

Alice finds out about Amazon Relational Database Service (RDS).

w00t! No more Alice the DBA!

But, there's a problem....

# Where is Alice's cozy Kubernetes experience?



Create database

Choose a database creation method Info

[✓ Standard Create] You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy Create: Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type Info

Amazon Aurora | MySQL | MariaDB | PostgreSQL | Oracle | [✓ Microsoft SQL Server]

## Alice doesn't have to use the AWS console...

What about the aws CLI tool?

What about CloudFormation?

What about Terraform?

## But those aren't Kubernetes.

And Alice loves her Kubernetes.

But not enough to be a DBA.

Alice wants RDS, but doesn't want to leave her Kubernetes experience.

What can Alice do?

## kubectl apply -f db.yml

```
apiVersion: rds.services.k8s.aws/v1alpha1
kind: DBInstance
metadata:
  name: mydb
spec:
  dbInstanceClass: db.m1.large
  dbInstanceIdentifier: mydb
  engine: PostgreSQL
…
```

# AWS Controllers for Kubernetes

Solving Alice's problems. And yours too, hopefully.

## A Kubernetes experience for AWS services

- Resources for AWS managed services are just another Kubernetes manifest
- Kubernetes stores the desired resource state
- ACK service controller handles the lifecycle of AWS managed service resources
- No CloudFormation behind the scenes

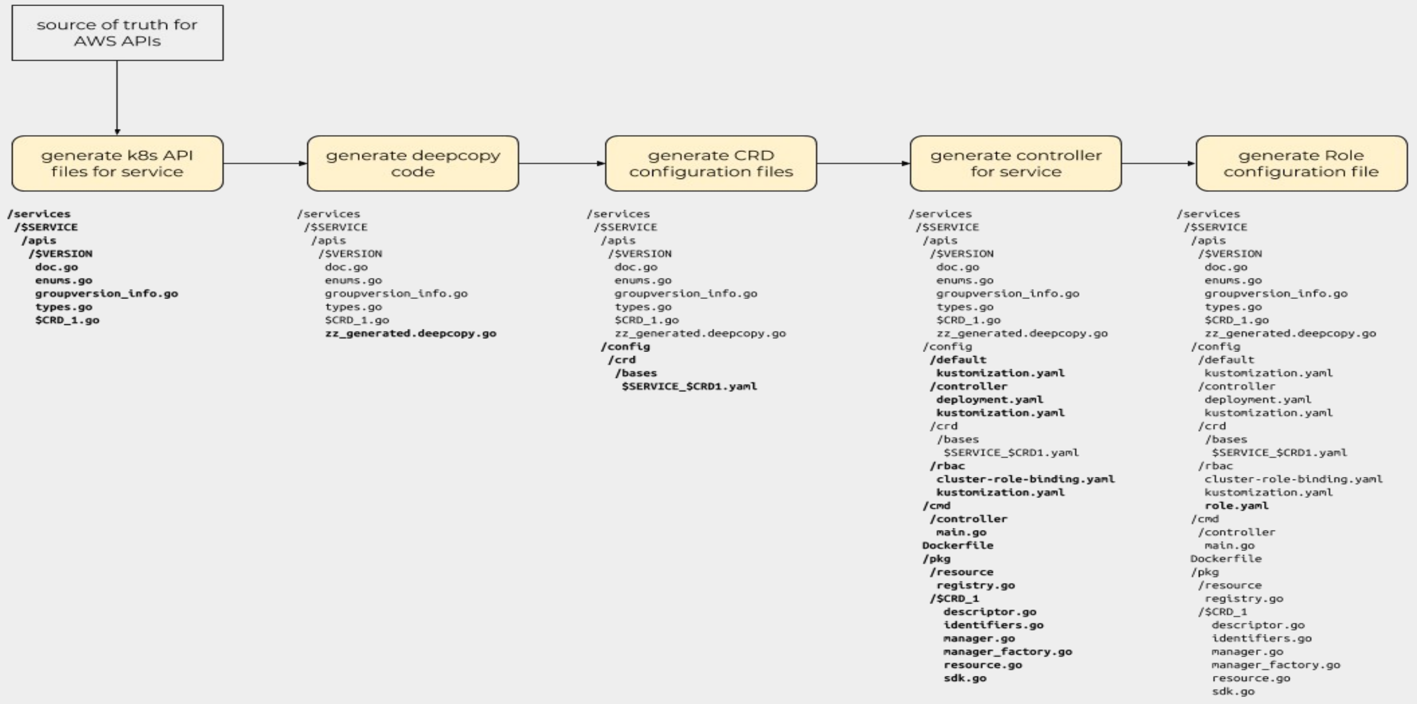
## Design of ACK

- Each AWS managed service has a separate ACK service controller
- Install using Helm, static manifests or helper script
- Everything, including controller implementation, is code-generated
- Consult with AWS service teams to ensure API calling and behaviour is correct
- Not EKS specific, runs on any Kubernetes

## Code generation

- Multiple phases
- Reads API models from aws-sdk-go, figures out which resources are CRDs (top-level)
- Creates Kubernetes API type definitions
- Generates resource manager for each CRD
- Generates linkage between AWS SDK and ACK runtime
- Generates config and build files

## Code generation


source of truth for AWS APIs
generate k8s API files for service
generate deepcopy code
generate CRD configuration files
generate controller for service
generate Role configuration file

```
/services
 /$SERVICE
  /apis
   /$VERSION
    doc.go
    enums.go
    groupversion_info.go
    types.go
    $CRD_1.go
```

```
/services
 /$SERVICE
  /apis
   /$VERSION
    doc.go
    enums.go
    groupversion_info.go
    types.go
    $CRD_1.go
    zz_generated.deepcopy.go
```

```
/services
 /$SERVICE
  /apis
   /$VERSION
    doc.go
    enums.go
    groupversion_info.go
    types.go
    $CRD_1.go
    zz_generated.deepcopy.go
  /config
   /crd
    /bases
     $SERVICE_$CRD1.yaml
```

```
/services
 /$SERVICE
  /apis
   /$VERSION
    doc.go
    enums.go
    groupversion_info.go
    types.go
    $CRD_1.go
    zz_generated.deepcopy.go
  /config
   /default
    kustomization.yaml
   /controller
    deployment.yaml
    kustomization.yaml
   /crd
    /bases
     $SERVICE_$CRD1.yaml
   /rbac
    cluster-role-binding.yaml
    kustomization.yaml
  /cmd
   /controller
    main.go
  Dockerfile
  /pkg
   /resource
    registry.go
   /$CRD_1
     descriptor.go
     identifiers.go
     manager.go
     manager_factory.go
     resource.go
     sdk.go
```

```
/services
 /$SERVICE
  /apis
   /$VERSION
    doc.go
    enums.go
    groupversion_info.go
    types.go
    $CRD_1.go
    zz_generated.deepcopy.go
  /config
   /default
    kustomization.yaml
   /controller
    deployment.yaml
    kustomization.yaml
   /crd
    /bases
     $SERVICE_$CRD1.yaml
   /rbac
    cluster-role-binding.yaml
    kustomization.yaml
    role.yaml
  /cmd
   /controller
    main.go
  Dockerfile
  /pkg
   /resource
    registry.go
   /$CRD_1
     descriptor.go
     identifiers.go
     manager.go
     manager_factory.go
     resource.go
     sdk.go
```

## Authorization and access control

- Kubernetes RBAC for custom resources (CRs)
- IAM Roles for AWS service and account permissions
- Each ACK service controller in own Deployment
- ACK service controller supplied with environment variables for AWS access credentials
- Use IAM Roles for Service Accounts to automate
- https://aws.github.io/aws-controllers-k8s/user-docs/authorization/

## Cross-account resource management (soon)

- Avoid installing ACK controllers in many clusters
- Kubernetes cluster admin associates an AWS account ID to a Namespace
- All ACK CRs must be in a Namespace
- Application developer creates AWS managed resources by creating CR in a Namespace
- ACK controller looks up Role ARN
- Controller calls STS::AssumeRole to pivot client

## What about secret stuff?

```
apiVersion: rds.services.k8s.aws/v1alpha1
kind: DBInstance
metadata:
  name: mydb
spec:
  dbInstanceClass: db.m1.large
  dbInstanceIdentifier: mydb
  engine: PostgreSQL
  masterUserPassword: UhmPlainText!?
...
```

## Ah, that's better. And more Kubernetes-like.

```yaml
apiVersion: rds.services.k8s.aws/v1alpha1
kind: DBInstance
metadata:
  name: mydb
spec:
  dbInstanceClass: db.m1.large
  dbInstanceIdentifier: mydb
  engine: PostgreSQL
  masterUserPassword:
    name: dbsecrets
    key: masterUserPassword
...
```

## Other things coming soon

- Standardized AWS tag representation for all ACK resources
- Control tags that all CRs (in a Namespace) should have
- Common rate limiting and throttling support
- “Adopting” pre-existing resources

## Developer preview

- S3 Bucket
- SNS Topic
- SQS Queue
- ECR Repository
- DynamoDB [Global]Table
- API Gateway V2

## Soon

- RDS DBInstance, DBCluster
- Elasticache CacheCluster
- CloudFront Distribution
- EC2 VPC Subnet, SecurityGroup, InternetGateway
- EKS
- Release roadmap:
  - https://github.com/aws/aws-controllers-k8s/projects/1

## Interested in contributing?

https://github.com/aws/aws-controllers-k8s

https://github.com/aws/containers-roadmap/issues/456