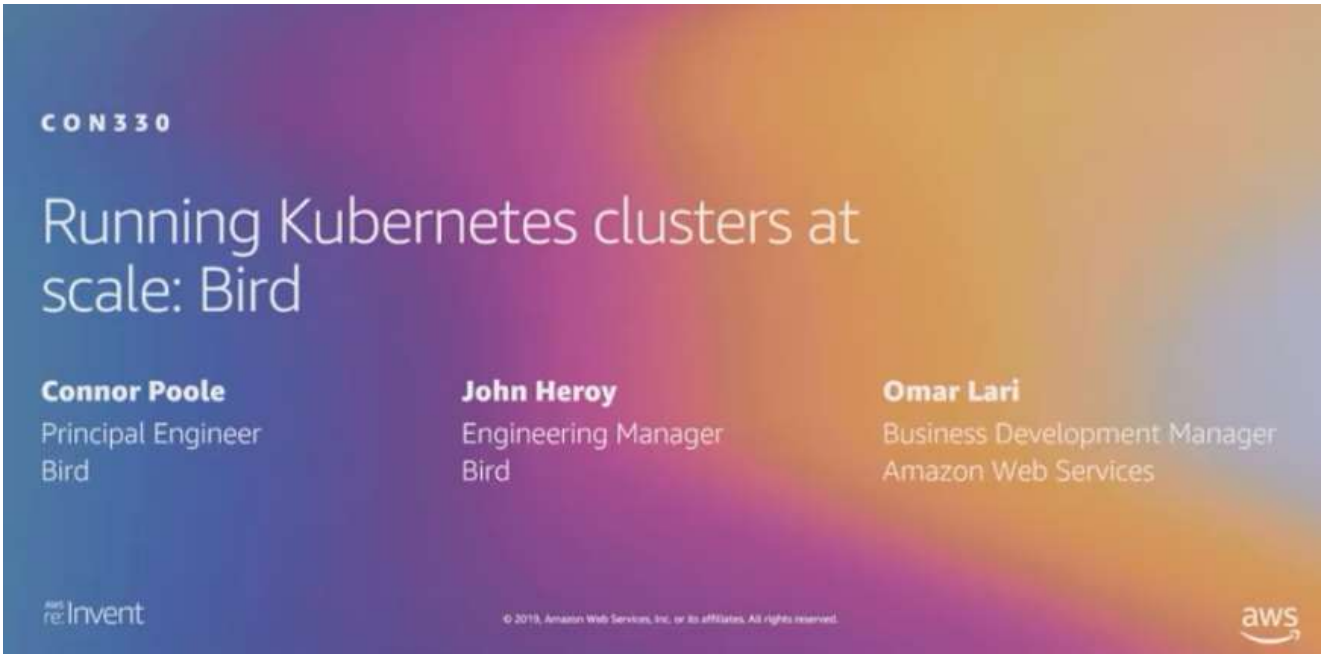
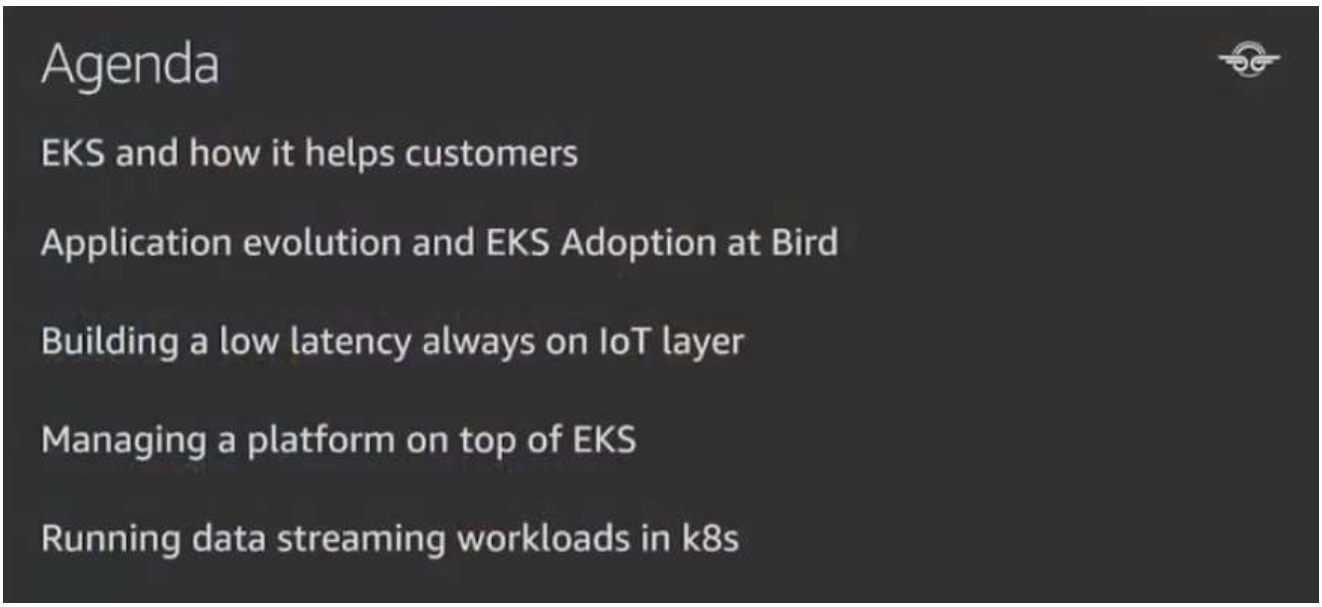


Launched in fall 2017, Bird is a micromobility company that enables access to shared e-scooters and lightweight electric vehicles in 100+ locations worldwide. Join us to hear how ***building a modern stack on top of Amazon EKS*** has enabled Bird to quickly ramp up its development in order to provide business value in a stable and secure manner. Further, learn how Bird's backend utilizes native AWS services like Amazon S3 and Amazon SQS, open-source streaming systems like Kafka and Flink, and a modern microservices architecture to turn terabytes of geospatial data into the mobility revolution of the future.



AWS best practices for building modern applications

- Create a culture of innovation by organizing into small DevOps teams
- Continually evaluate your security posture by automating security
- Componentize applications using microservices
- Update applications & infrastructure quickly by automating CI/CD
- Standardize and automate operations by modeling infrastructure as code
- Simplify infrastructure management with serverless technologies
- Improve application performance by increasing observability

What is Kubernetes?

Open source container management platform

Helps you run containers at scale

Gives you primitives for building modern applications

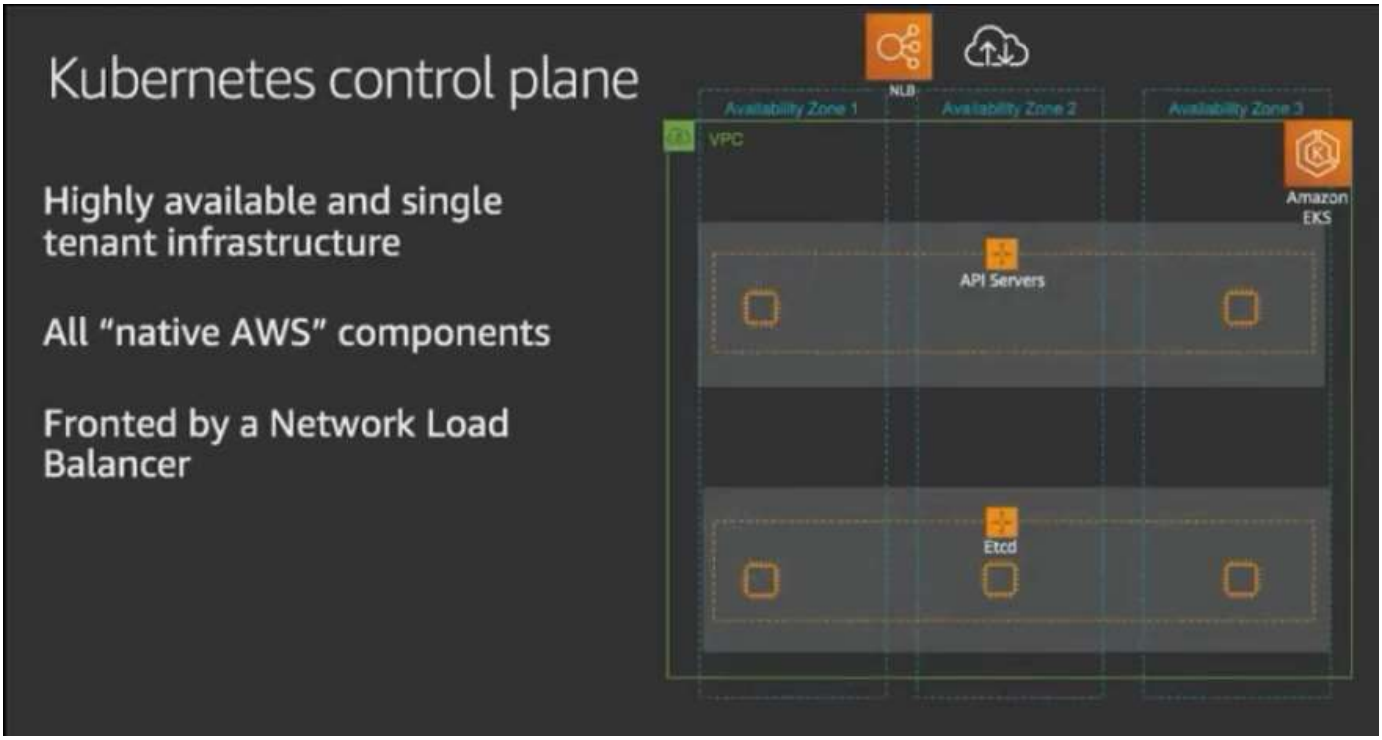
How are customers using Amazon EKS?

Microservices

Platform as a service

Enterprise App Migration

Machine Learning



This is what you get when you run a '$ eks create cluster' command in your account, you get a HA ectd cluster to store cluster data and a HA k8s control plane (the API Servers, Controller Managers, etc.) that gets exposed to you via a Network LB to connect you client tooling like kubectl or your other apps. These are all encrypted.

Amazon EKS, a year in review

June – December 2018:

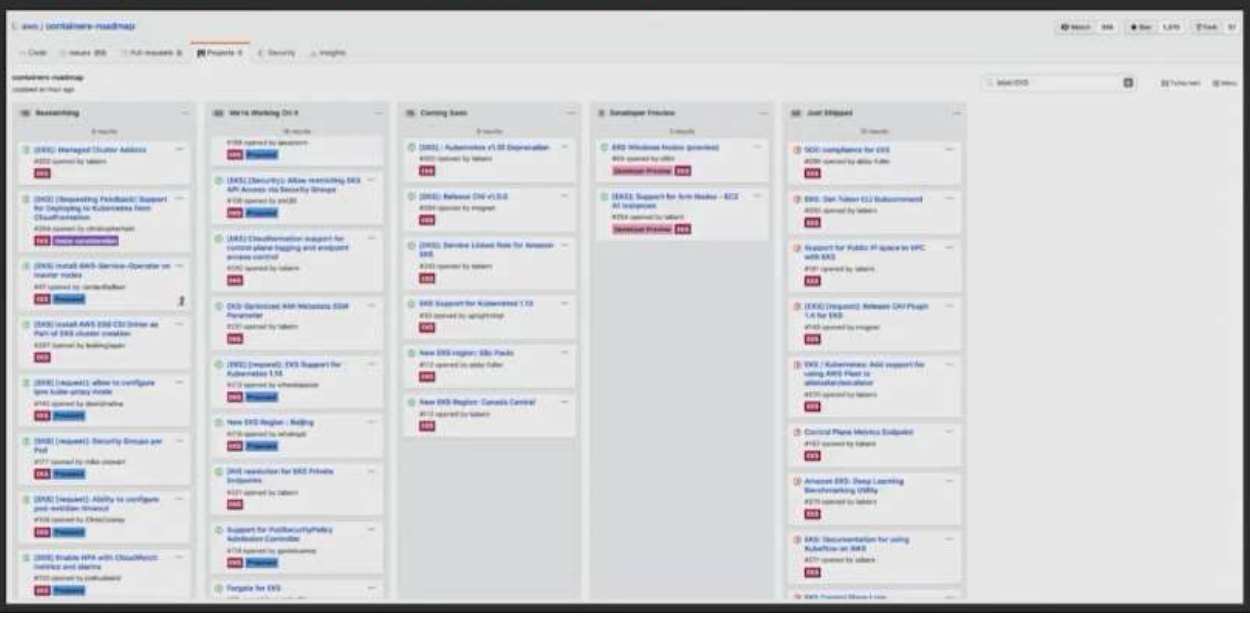
Amazon EKS achieves K8s conformance, HIPAA-eligibility, Generally available
Amazon EKS AMI build scripts and AWS CloudFormation templates available in GitHub.
Support for GPU-enabled EC2 instances, support for HPA with custom metrics.
Amazon EKS launches in Dublin, Ireland
Amazon EKS simplifies cluster setup with `update-kubeconfig` CLI command
Amazon EKS adds support for Dynamic Admission Controllers (Istio), ALB Support with the AWS ALB ingress controller
Amazon EKS launches in Ohio, Frankfurt, Singapore, Sydney, and Tokyo
Amazon EKS adds Managed Cluster Updates and Support for Kubernetes Version 1.11, CSI Driver for Amazon EBS

2019:

Amazon EKS launches in Seoul, Mumbai, London, and Paris
Amazon EKS achieves ISO and PCI compliance, announces 99.9% SLA, cluster creation limit raised to 50
API Server Endpoint Access Control, AWS App Mesh controller
Windows support (preview), Kubernetes version 1.12,
CSI Drivers for Amazon EFS, Amazon FSx for Lustre, Control Plane Logs, A1 (ARM) instance support (preview)
Deep Learning Benchmark Utility, Public IP Address Support,
Simplified cluster authentication, SOC compliance, Kubernetes 1.13, PodSecurityPolicies,
Container Insights, CNI 1.5.0, Amazon ECR, AWS PrivateLink Support
Managed Nodes

Open-source roadmap

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



Amazon EKS services roadmap: Highlights

Shipped

- IAM Roles for Service Accounts (pods)
- Managed Node Groups
- K8s version 1.13, 1.14
- ECR Private Link
- New Amazon EKS Regions: Paris, London, Mumbai, Hong Kong, Sau Paulo, Montreal

Coming soon

- Fargate for EKS
- Service Linked Role
- KMS Encryption Provider
- New Amazon EKS Regions: Beijing, Ningxia

Working on it

- Managed add-ons
- DNS resolution of Amazon EKS private endpoints
- New Amazon EKS Regions
- Next-generation CNI plugin

BIRD

Building a platform on a deadline

Reduce, reuse, recycle

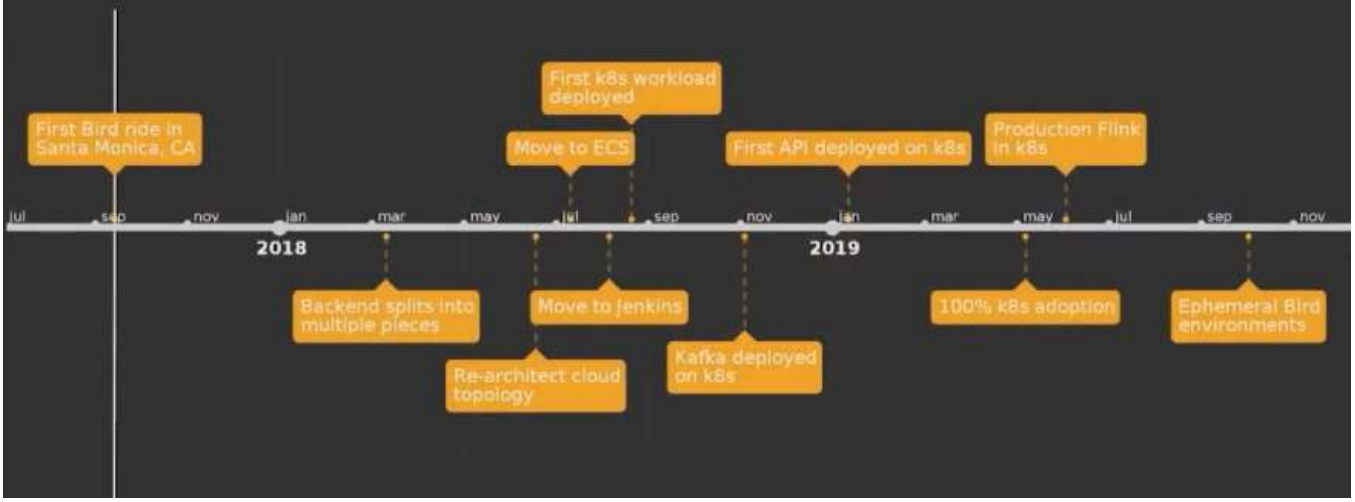
Infrastructure: Terraform

CI/CD: Jenkins

Orchestration: Amazon EKS

Managed Services: AWS

Timeline



September 2017

- First Bird ride in Santa Monica
  - Modified retail scooters
  - Phone apps and a single backend

March 2018

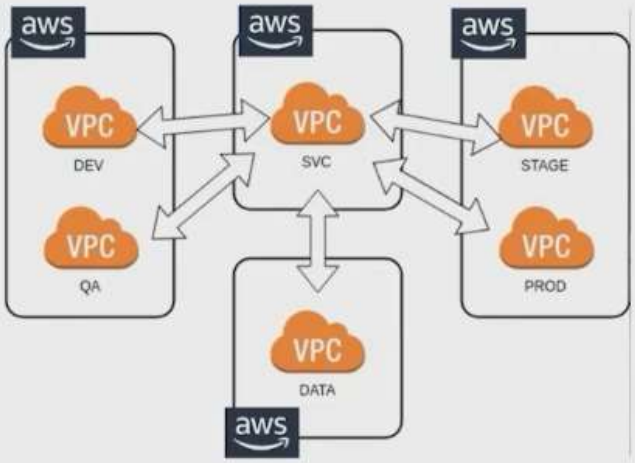
- Second deployable emerges
  - Fleet size has grown considerably
    - Thousands of Birds
    - Multiple markets in the US
    - Began decoupling batch processes into "workers"

June 2018

- Start fresh in AWS
  - Re:Invent the wheel

Cloud topology

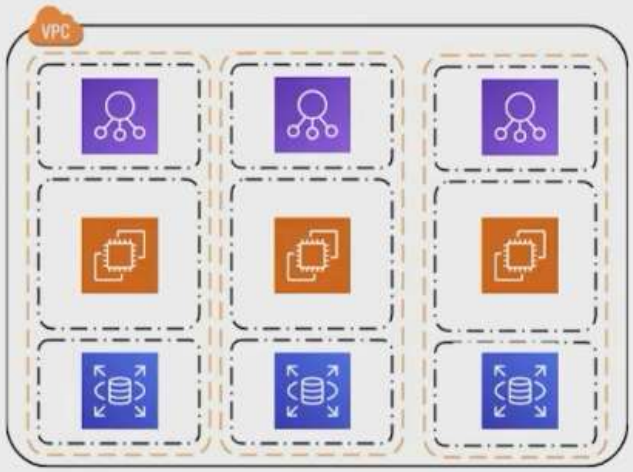
- Hub-and-spoke topology
- Accounts are mostly 1:1 with spokes



VPC design

Standard three-level VPC design

Public, private, Fort Knox



Roll with the punches, and embrace detours

- Outages frequently
- Deploying jars from Amazon S3 wasn't working for us
- Multiple environments + CI environments needed more encapsulation
- Amazon EKS was still in beta at this time, so we migrated fully to Amazon Elastic Container Service (Amazon ECS) in under 30 days

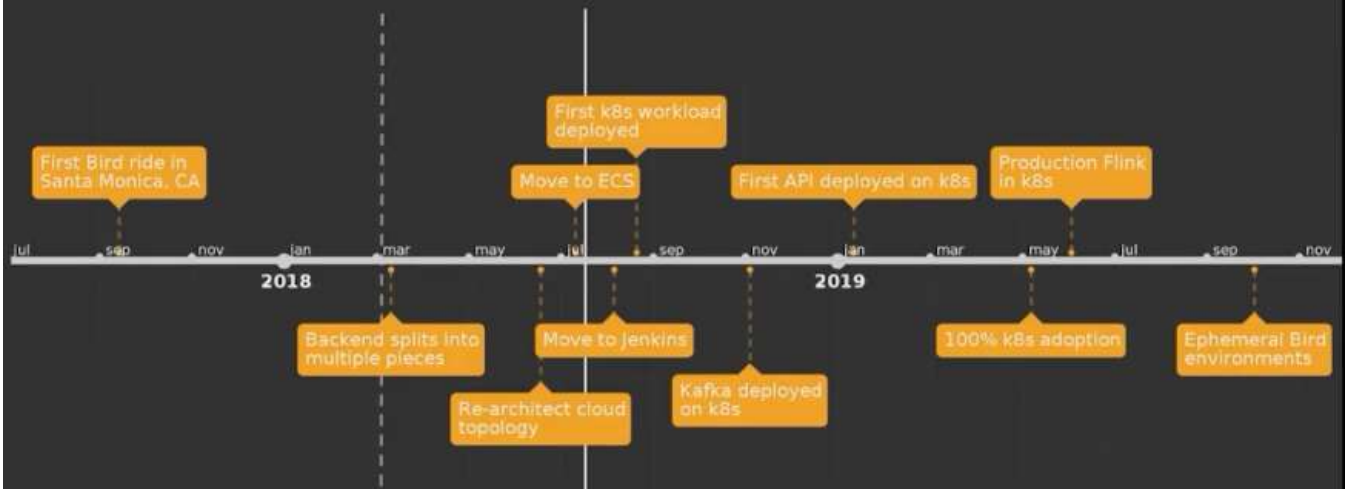
Why Amazon ECS?

We needed to get onto containers ASAP

Amazon ECS is simple to set up and had native AWS Identity and Access Management (IAM) integration

Was a smaller jump from Amazon EC2 to Amazon EKS

Timeline recap



Why Amazon EKS?

Supports advanced stateful workloads

Amazon EKS removes all the headaches of k8s control plane management

Native VPC integration

Managing Amazon EKS

- Prune your clusters to a known state on startup
- Use large workers for bin packing
- Rely on Cluster Autoscaler + Horizontal Pod Autoscaler
- Calico for network policy

Who orchestrates the orchestrator?

```
module "eks" {
  vpc_id             = module.vpc.id
  security_group_ids = [ … ]

  efs_security_group = module.vpc.aws_security_group-allow_all_within_vpc.id
  efs_subnets        = module.vpc.private_subnets.*.id

  subnet_ids = concat(module.vpc.public_subnets.*.id,module.vpc.private_subnets.*.id)

  name_prefix         = var.eks_cluster_prefix

  private_ingress_ssl_cert        = aws_acm_certificate.cert.arn
  private_ingress_security_groups = module.vpc.aws_security_group-vpc_local.id
  public_ingress_ssl_cert         = aws_acm_certificate.cert.arn
  public_ingress_security_groups  = module.vpc.aws_security_group-extenal_access.id

  vault_auth_enabled = 1
}
```

We use Terraform to manage our K8s clusters across our account, this is what a module looks like in 20 lines

Partitioning a cluster

- Cluster sprawl is real
- Namespaces help clamp the chaos
- Choose your divisions wisely; too many namespaces is equally bad

Use namespaces to make pods and items naming and location easy

Choose a k8s deployment strategy

- Minimize cruft by handling all object creation in a unified manner
- We chose Helm for its flexibility and community support
- Deployment history through configmaps provides a clean API to code against

Use I deployment strategy, we use Helm because it provides a clean API that we can code against and work with

Except you are doing it with K8s on EKS

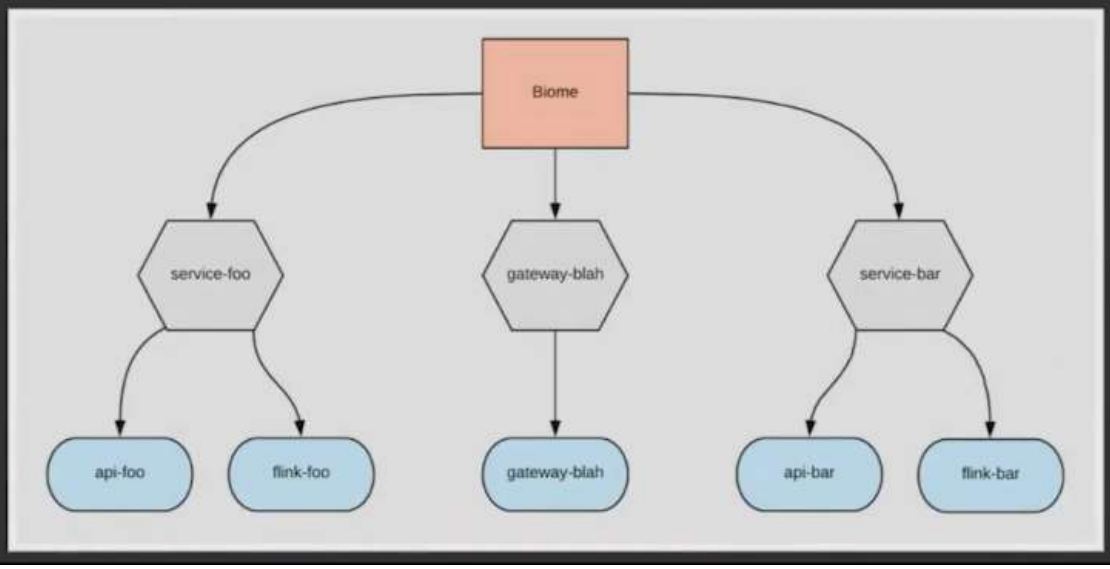
Data pipeline begins

- 216 nodes per cluster
- 500 GB per disk
- 10 Kafka clusters per EKS cluster
- Be prepared for disk space resizing at 11 p.m. on your laptop, pulled over to the side of the road

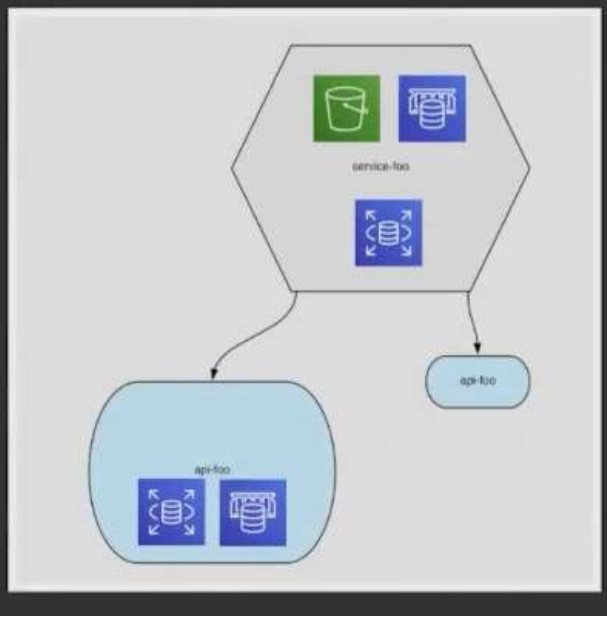
This is the first workflow we deployed on EKS, we run multiple Kafka clusters on EKS

Monolith decomposition begins

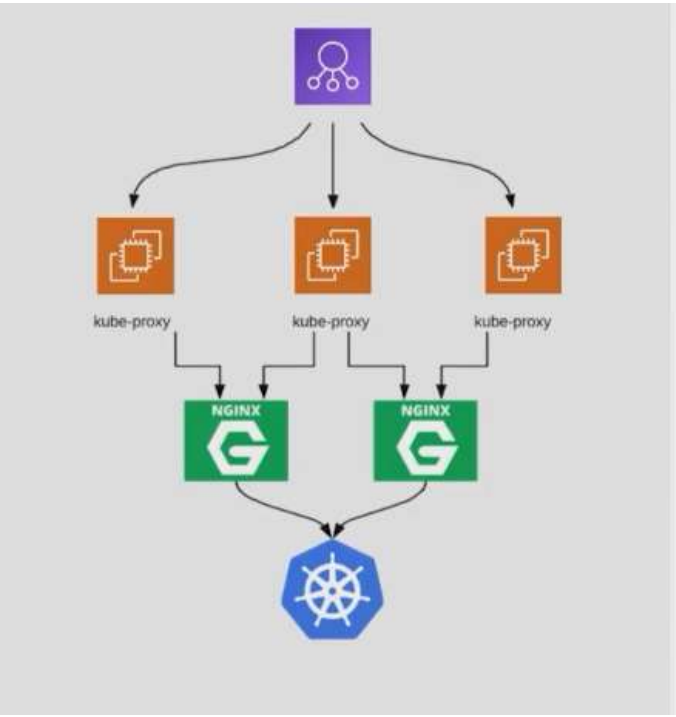
- Organize your apps, data, and configs in a standard structure



Colocate your data sources



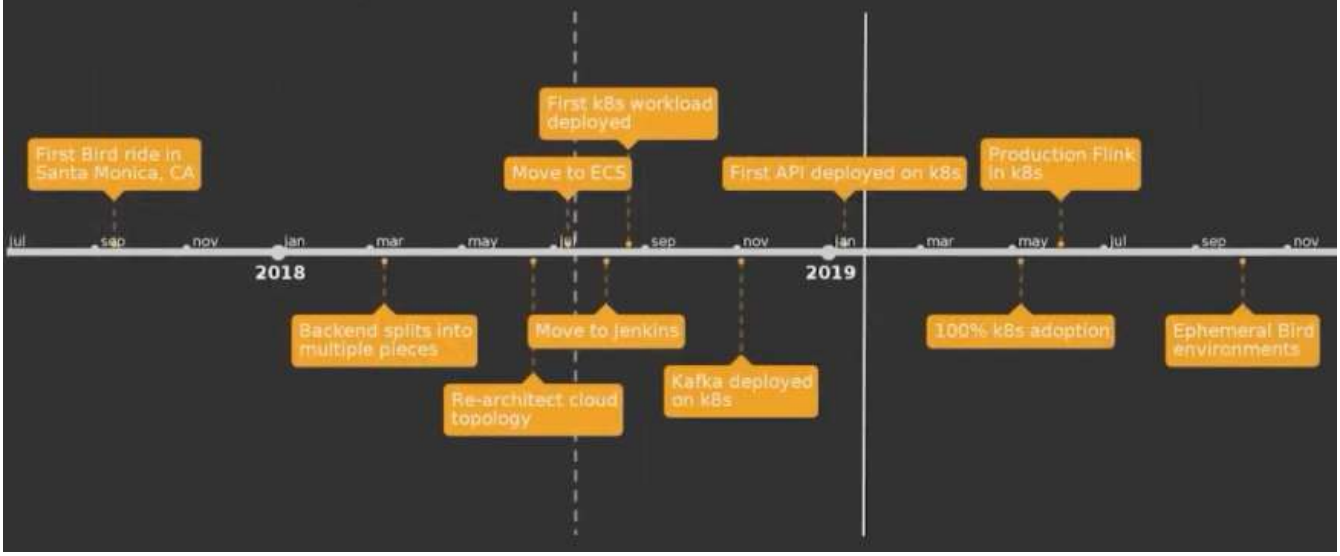
Use namespace properly to grant restricted access to teams



To mesh or not to mesh

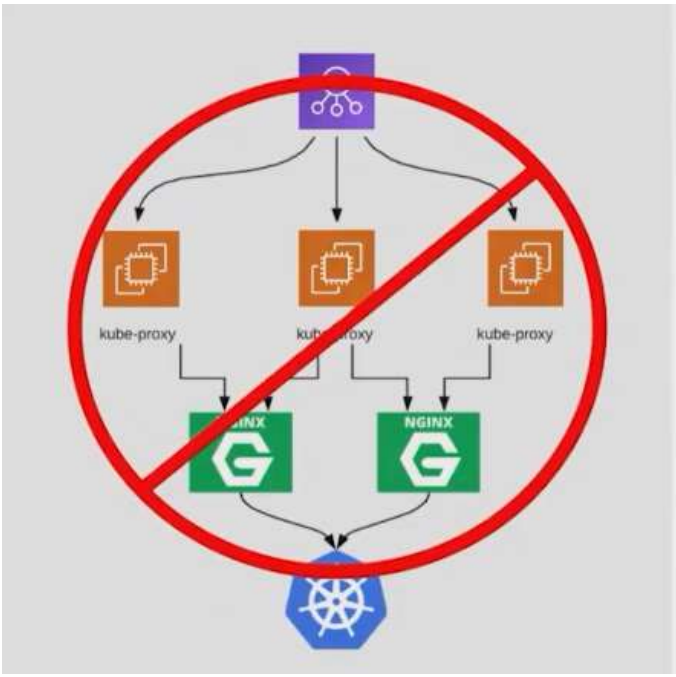
- Do you really need a full mesh?
- Ingress-nginx was good enough for us
- It is simple to set up, scale, and run locally

Timeline recap



Let's talk about vehicles

- Bird has built a large network of always-connected IoT devices
- The backend should always be able to connect to our vehicles

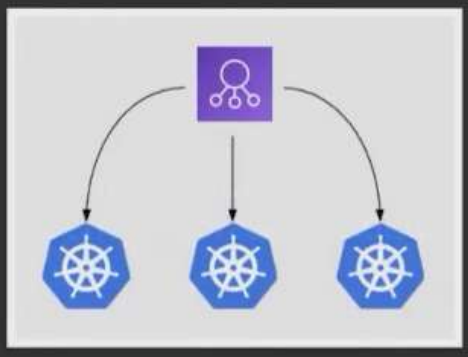


Controlling TCP

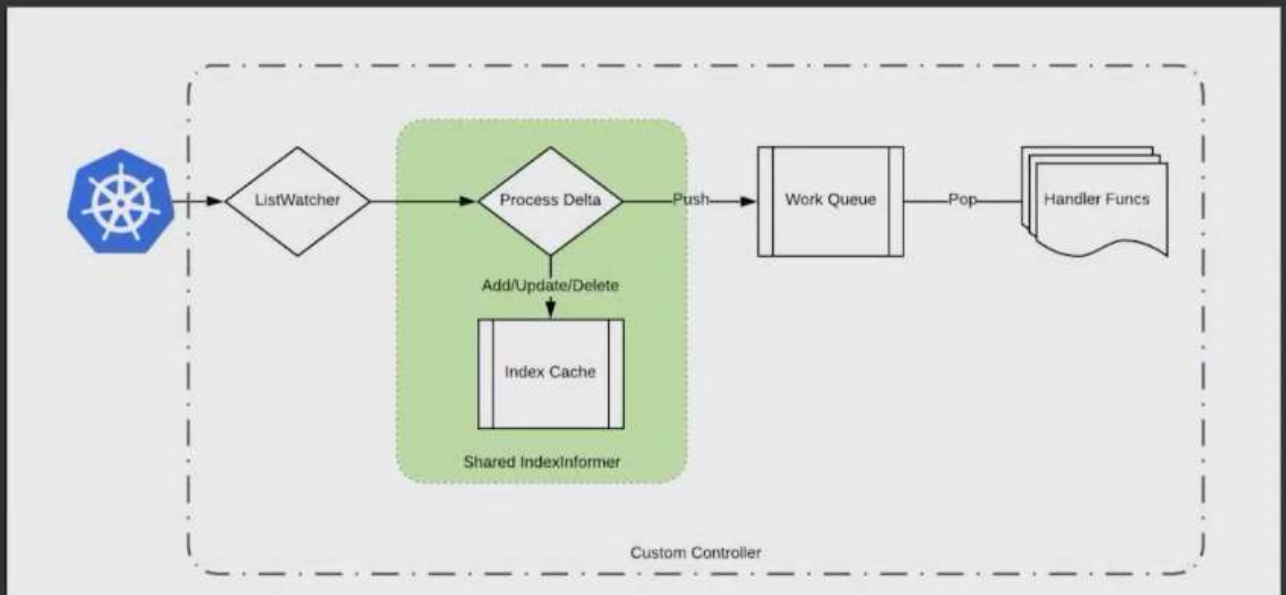
- We needed to ensure infinite socket life for our IoT gateway
- NLB integration with Kubernetes wasn't enough

NLB attacher

- Direct from NLB to pods
- Lower latency, fewer moving pieces
- Most important – infinite socket life



Building a k8s controller



We extended our K8s with custom controllers

Core controller

```go
// Controller - the primary struct responsible for all cluster actions
type Controller struct {
    clientset       kubernetes.Interface
    queue           workqueue.RateLimitingInterface
    informer        cache.SharedIndexInformer
    eventHandler    handlers.Handler
    config          config.Config
    serverStartTime time.Time
}

const maxRetries int = 5
const enableLabelValue string = "nlb-attacher.bird.co/enabled"
const targetGroupAnnotationKey string = "nlb-attacher.bird.co/target-groups"
```

Handler interface

```go
// Handler is implemented by any handler.
// The Handle method is used to process event
type Handler interface {
    Init(tgAnnotation string, annotationEnabledValue string) error
    PodCreated(created *v1.Pod)
    PodDeleted(deleted *v1.Pod)
    PodUpdated(oldPod, newPod *v1.Pod)
    TestHandler()
}
```

Adding pods

```go
// PodCreated - Handle the creation event of a pod and ensure it's attached to all of the specified target groups
func (handler *Handler) PodCreated(created *v1.Pod) {
    if created.DeletionTimestamp != nil {
        log.Infof("Discovered pod %s with deletionTimestamp already set. skiping...", created.Name)
        return
    }

    if created.Status.PodIP == "" {
        log.Infof("Recieved event for pod %s without an ip address yet. skipping...", created.Name)
        return
    }

    log.Debugf("created object: %v", created.Name)
    handler.addToTargetGroups(created)
}
```

From Bird with love

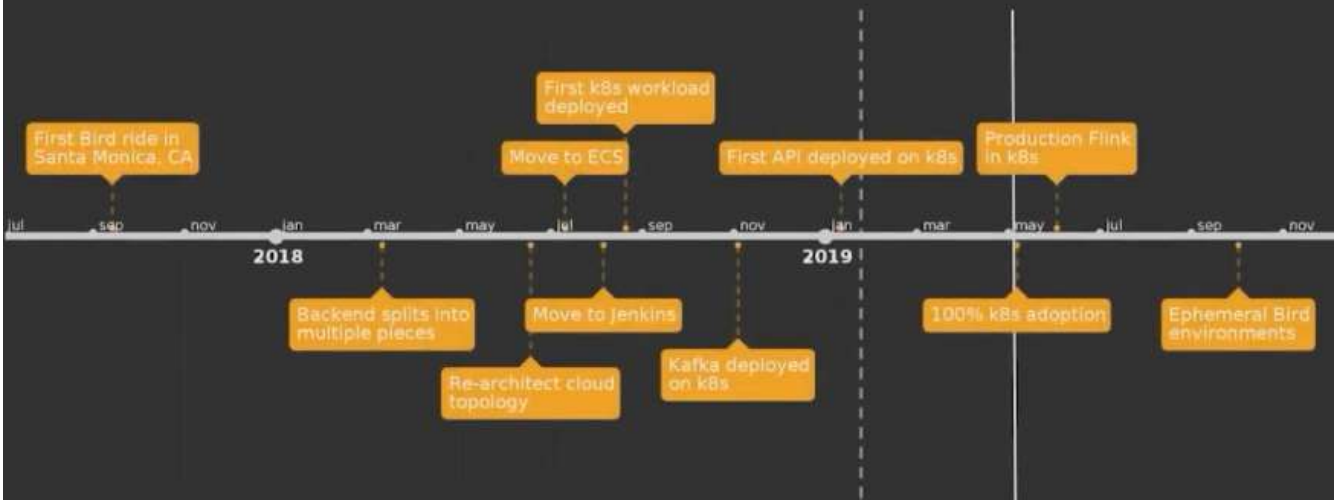
Open-sourcing this tool today:

https://github.com/birdrides/nlb-attacher

Future roadmap:

- Pod UDP support
- Target group creation

Timeline recap



Clamping state drift

- You're never going to fully solve this
- Carrot, not the stick
- Centralize all deployment pieces in as few graphs as possible
- Consider using CRDs to represent as many pieces of critical infrastructure as possible

But be realistic

- Terraform is still our favorite tool today
  - A single graph can combine Helm, Kubernetes, Vault, CDN providers, etc.
- The interface between Helm and Terraform is poor
  - A brittle, weakly typed, whitespace-sensitive contract will cause headaches
  - Perform your logic inside common Helm templating libraries or inside Terraform modules that wrap Helm
- Look to the future:
  - https://github.com/aws/aws-service-operator-k8s
  - https://github.com/rancher/terraform-controller

A platform is born

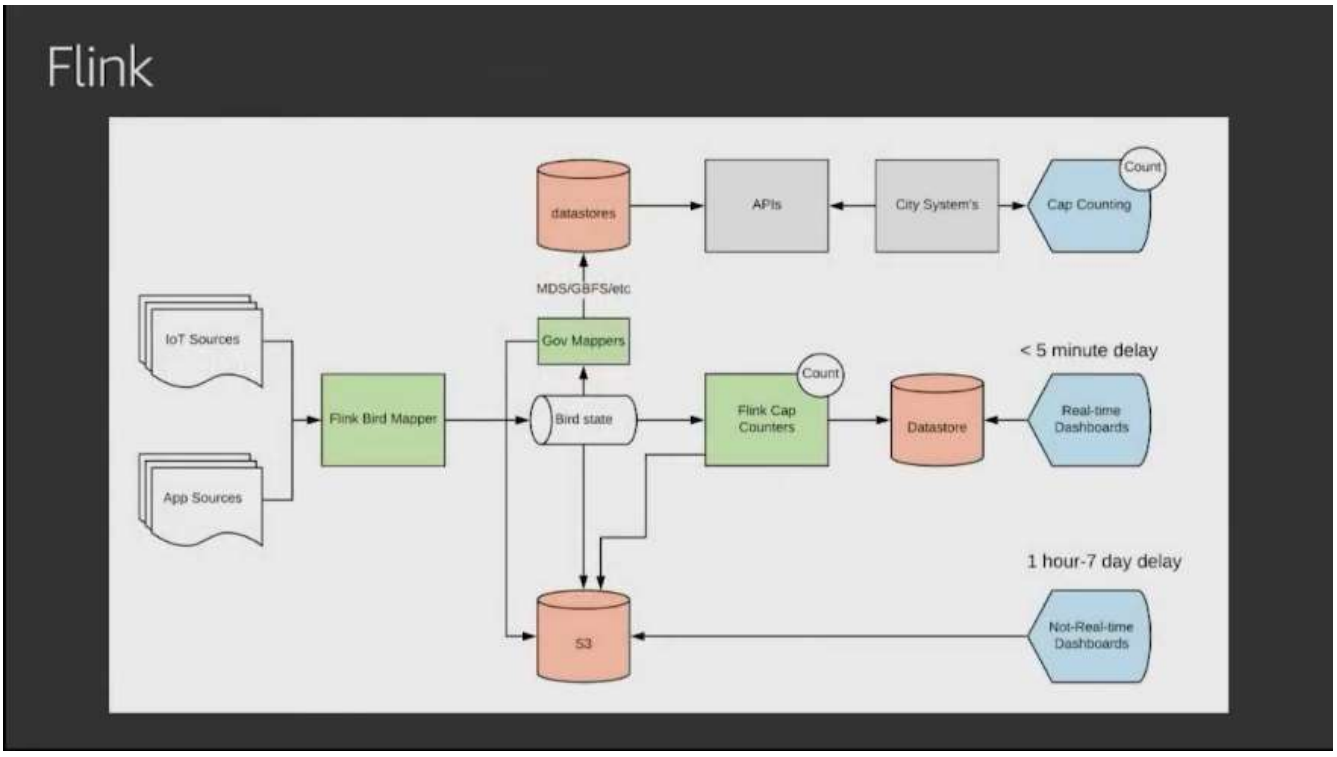
- Understand your users
  - "DevOps" is a cultural goal in moderation
- Kubernetes evolves quickly
  - Abstraction is your friend – keep your abstraction
    - Keep your abstraction thin – shim cloud native
- Tighten the feedback loop
  - Instant gratification is key
    - Users must see results/feedback during their testing loops

Interacting with our platform

```
{
  "name": "service-habitat",
  "dependencies": [
    "sqs",
    "postgres"
  ],
  "deployables": [
    {
      "name": "api-habitat",
      "dependencies": [

      ],
      "language": {
        "name": "kotlin",
        "properties": {
          "framework": "vertx",
          "main_class": "co.bird.habitat.api.HabitatVerticle"
        }
      },
      "type": {
        "name": "http-api"
      }
    }
  ]
}
```

This is a Bird service that we abstracted



But why Flink?

We need data in real time

State transitions are infrequent but latency-sensitive

Native k8s support

Too many vehicles for batch

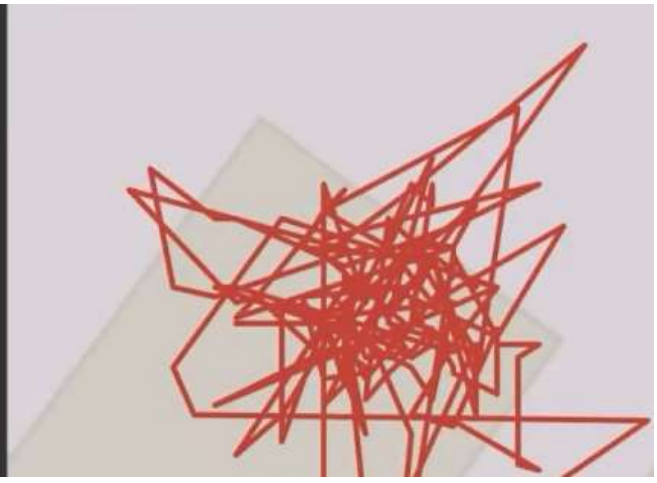
Best in memory state management, snapshot, and debugging tooling

Why not Flink?

- Flink is still new, bugs exist, and we regularly find them
- Kafka + Flink have silent partition issues
- You're not already on k8s

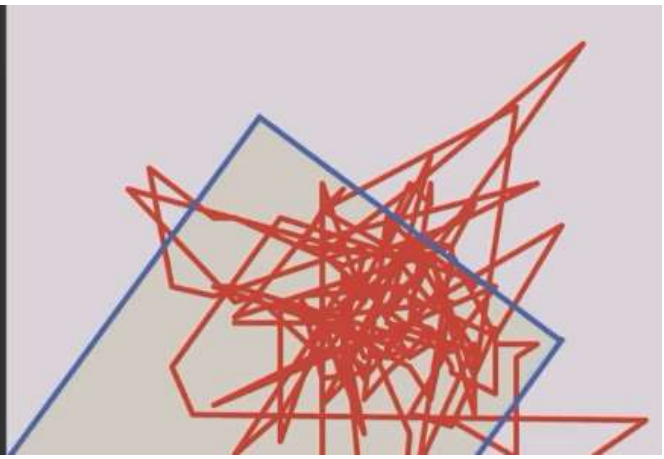
IoT signals are messy

Where do you think this Bird is?



IoT signals are messy

The Bird is in a superposition of states!



Accurate signals are critical

Location:

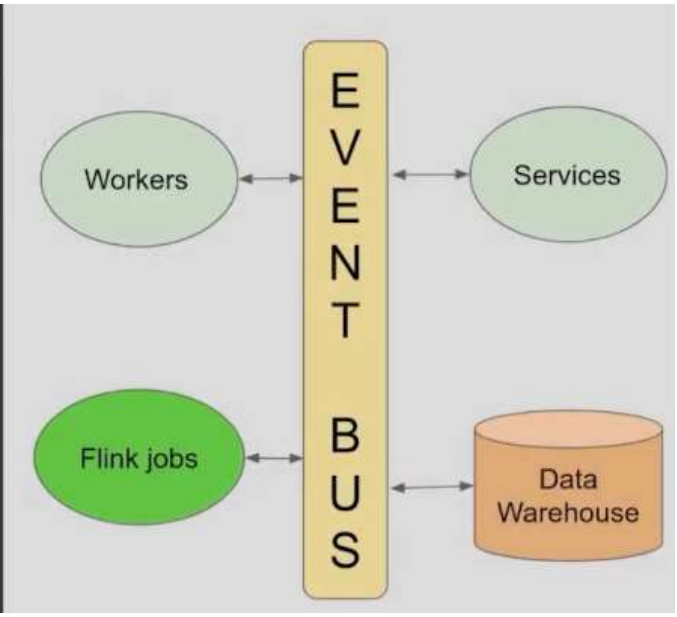
- Within polygons
- Distance from other POI

Stream processing is flexible

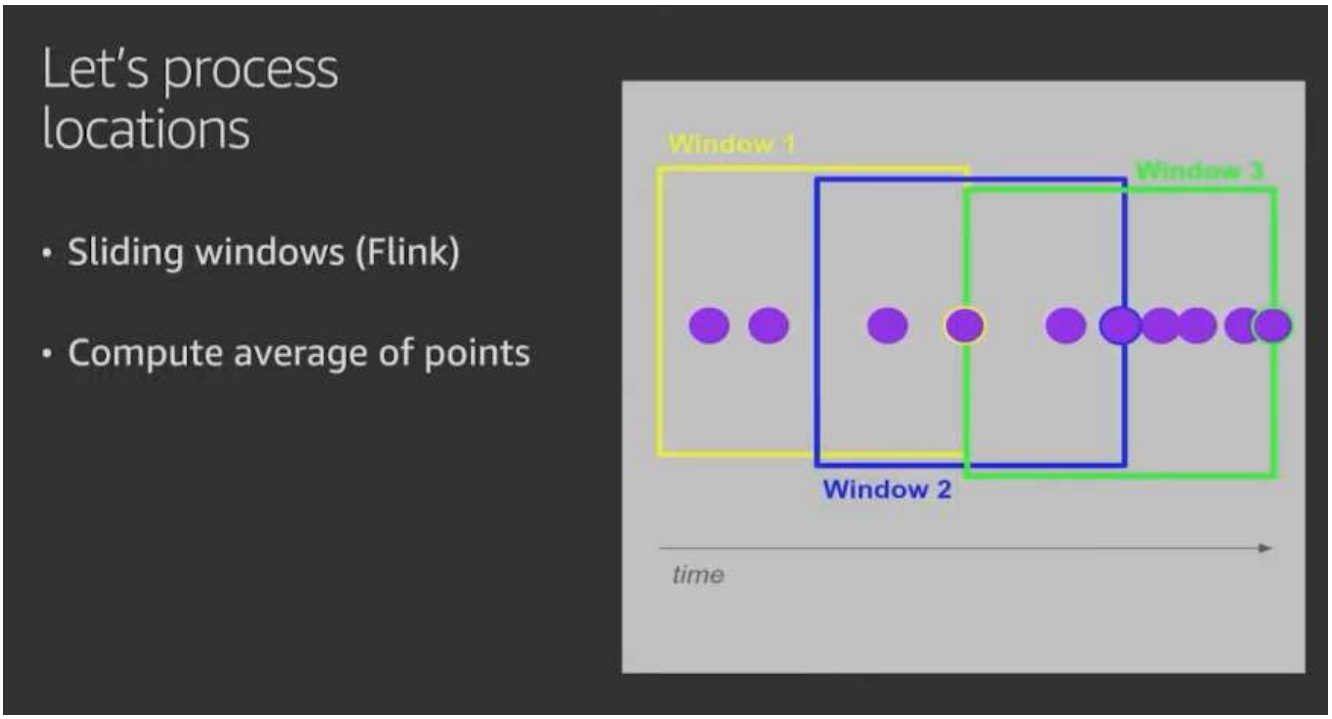
Quickly grow our ecosystem

- Events
- Publishers
- Consumers

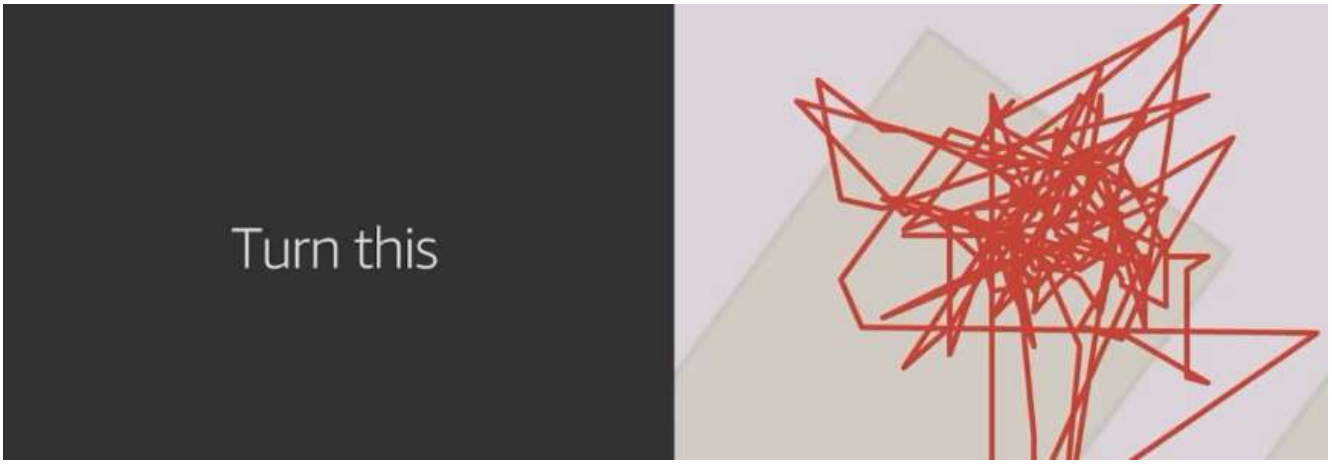
Derived and raw events

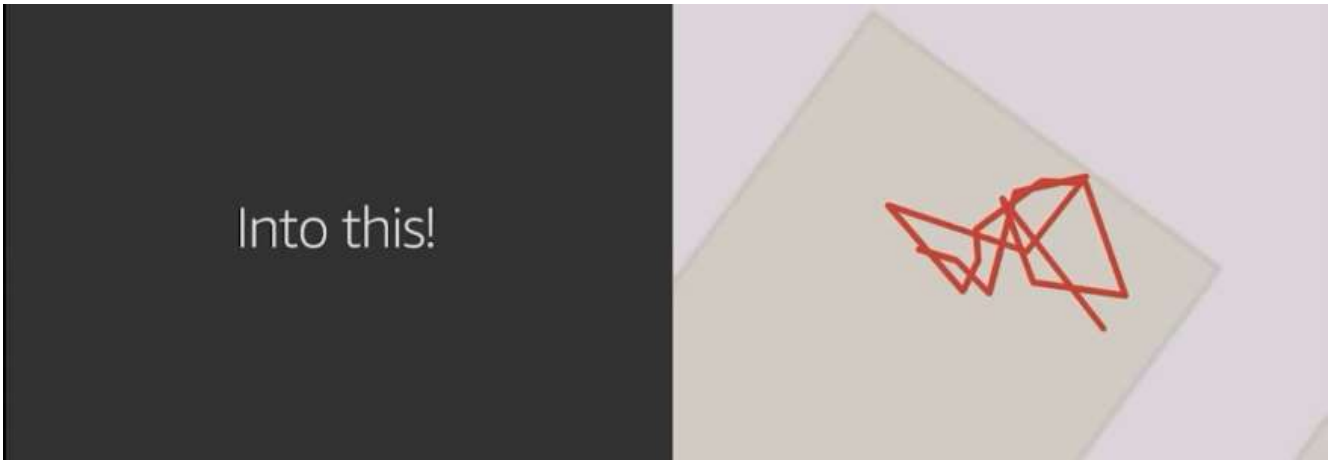


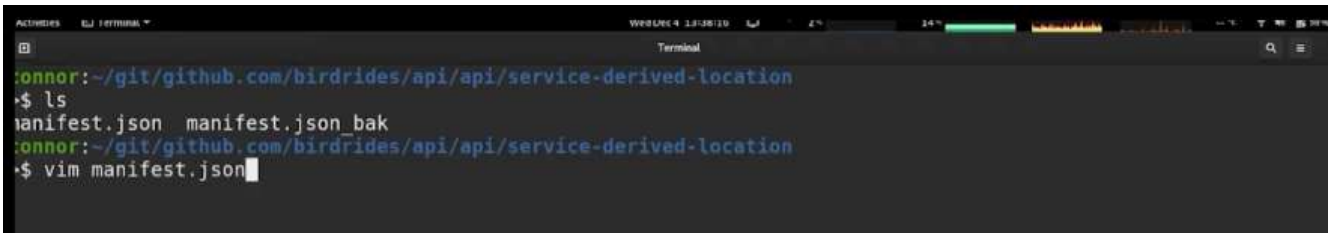
Kafka is our global event bus

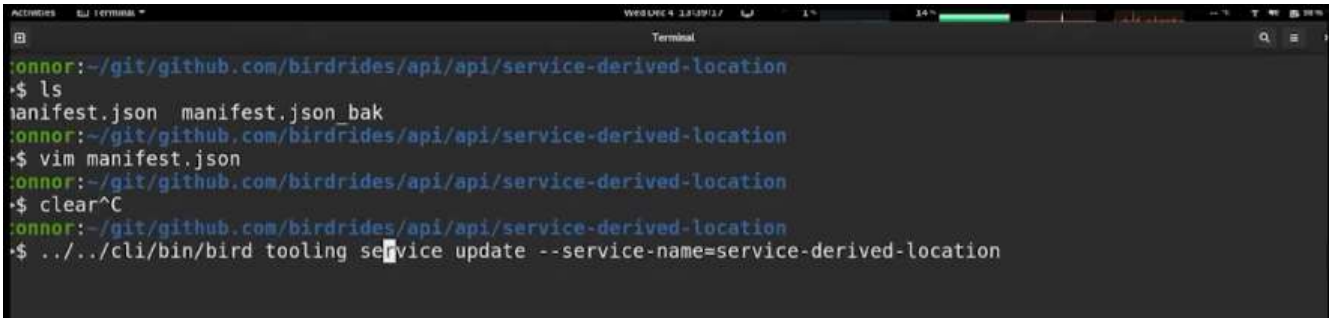


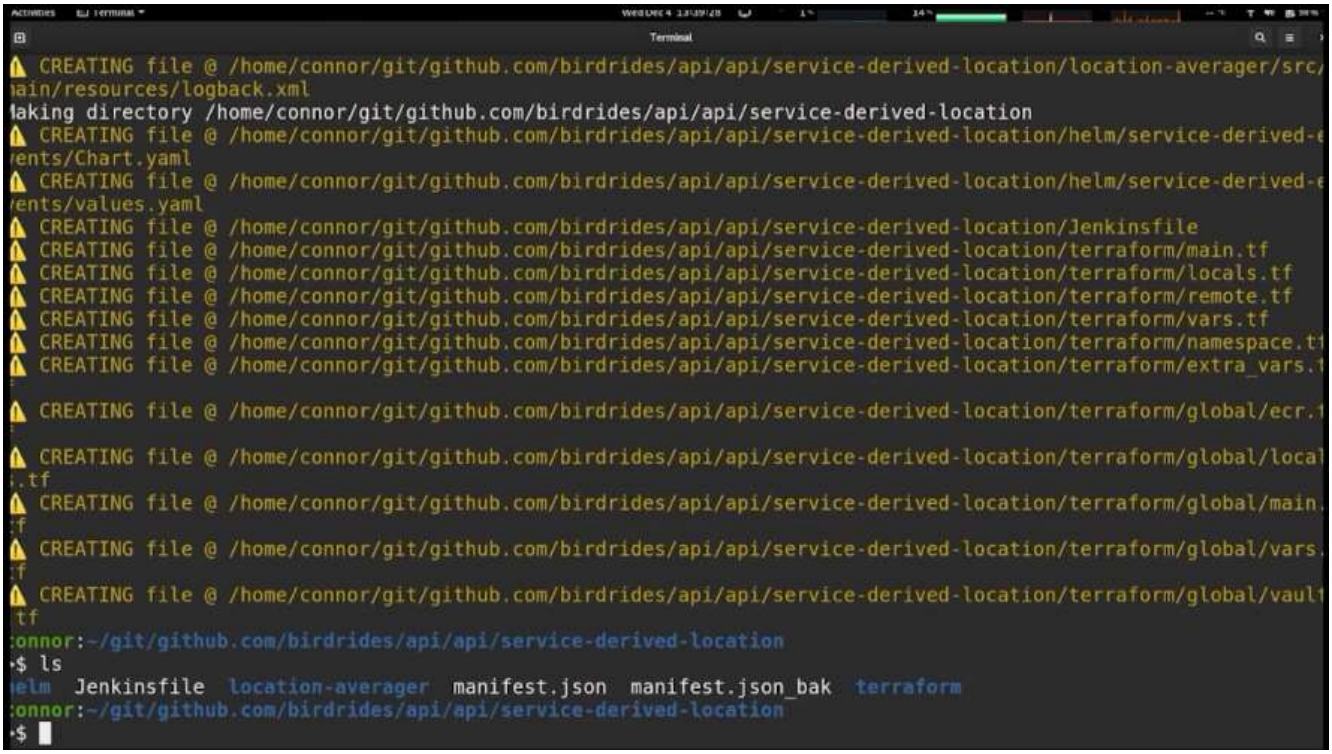
We declare a Flink Job



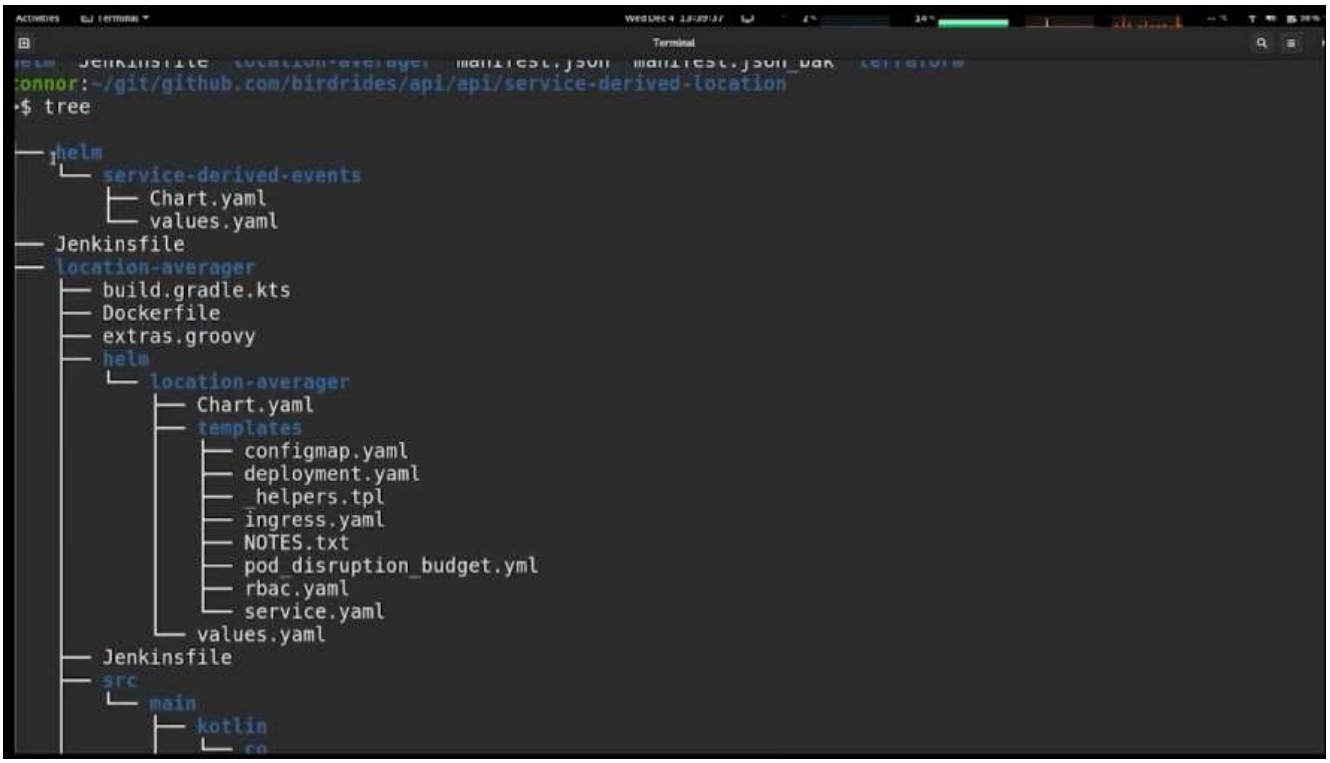


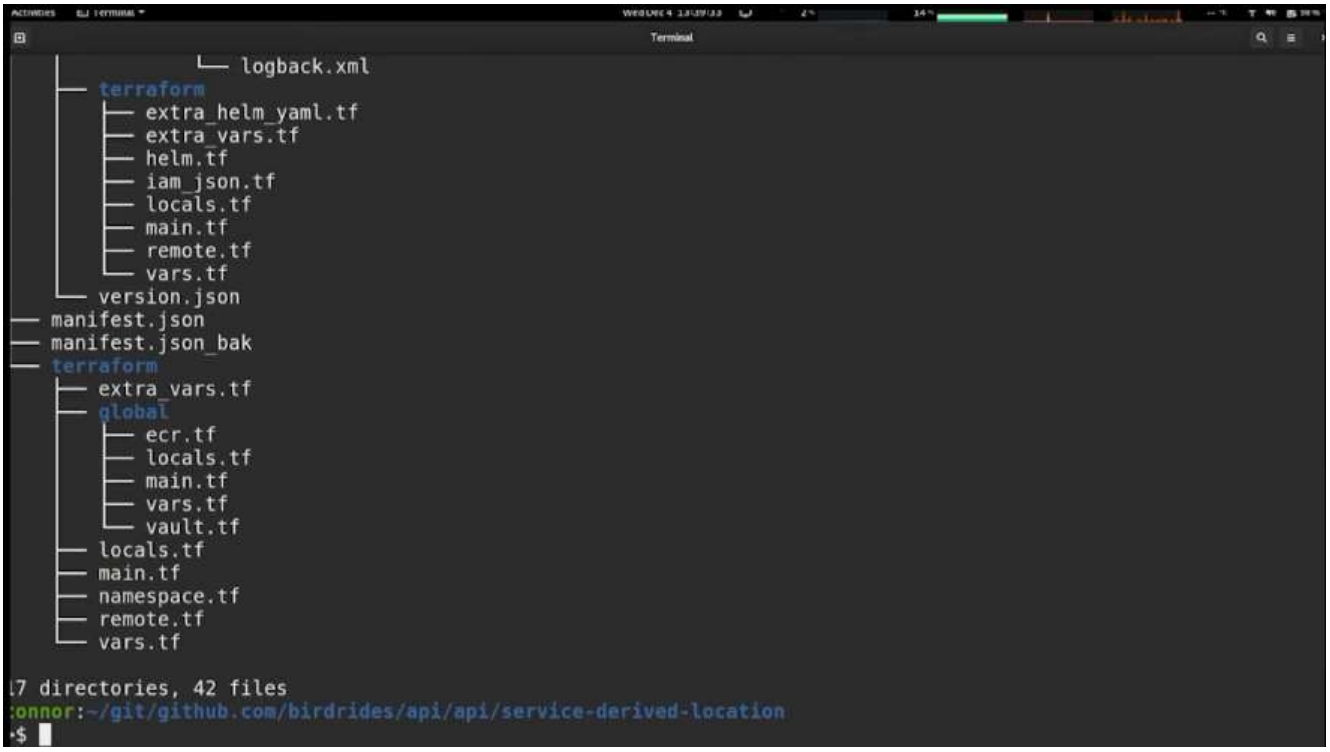


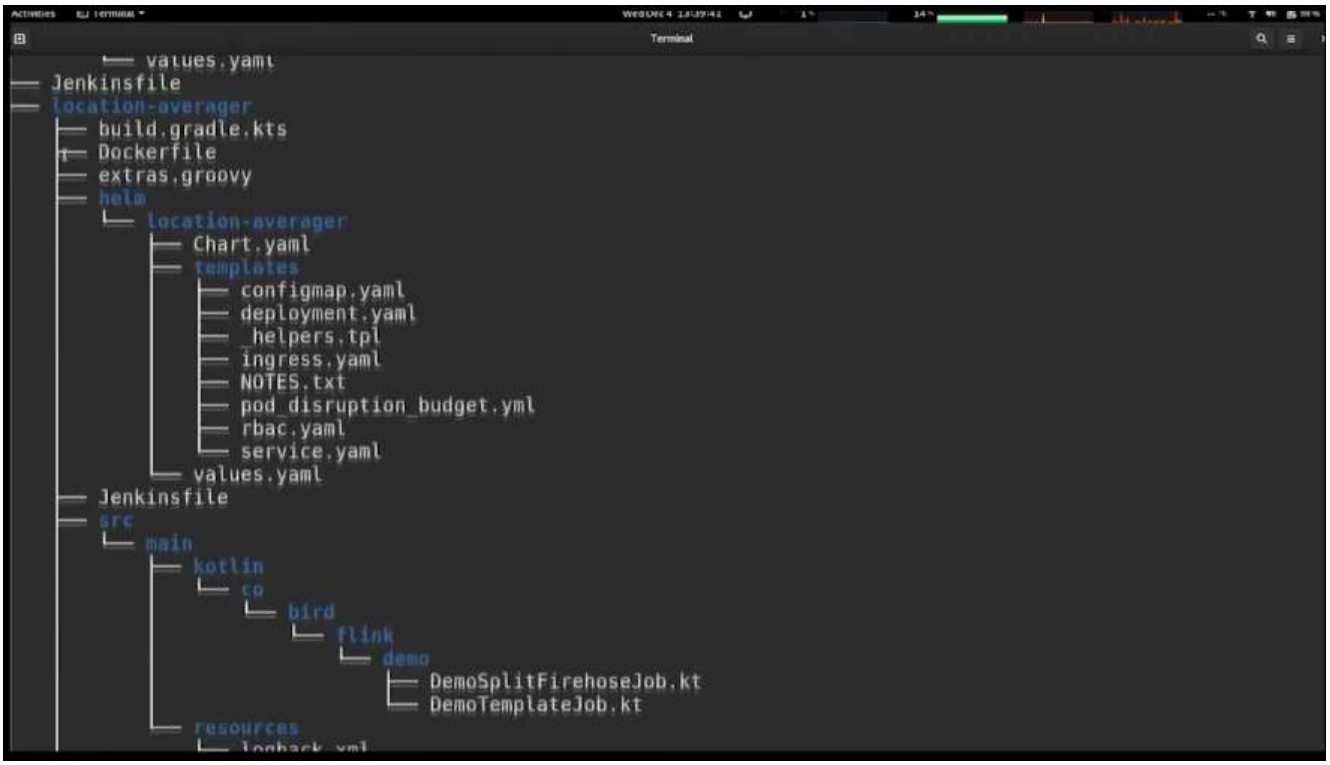


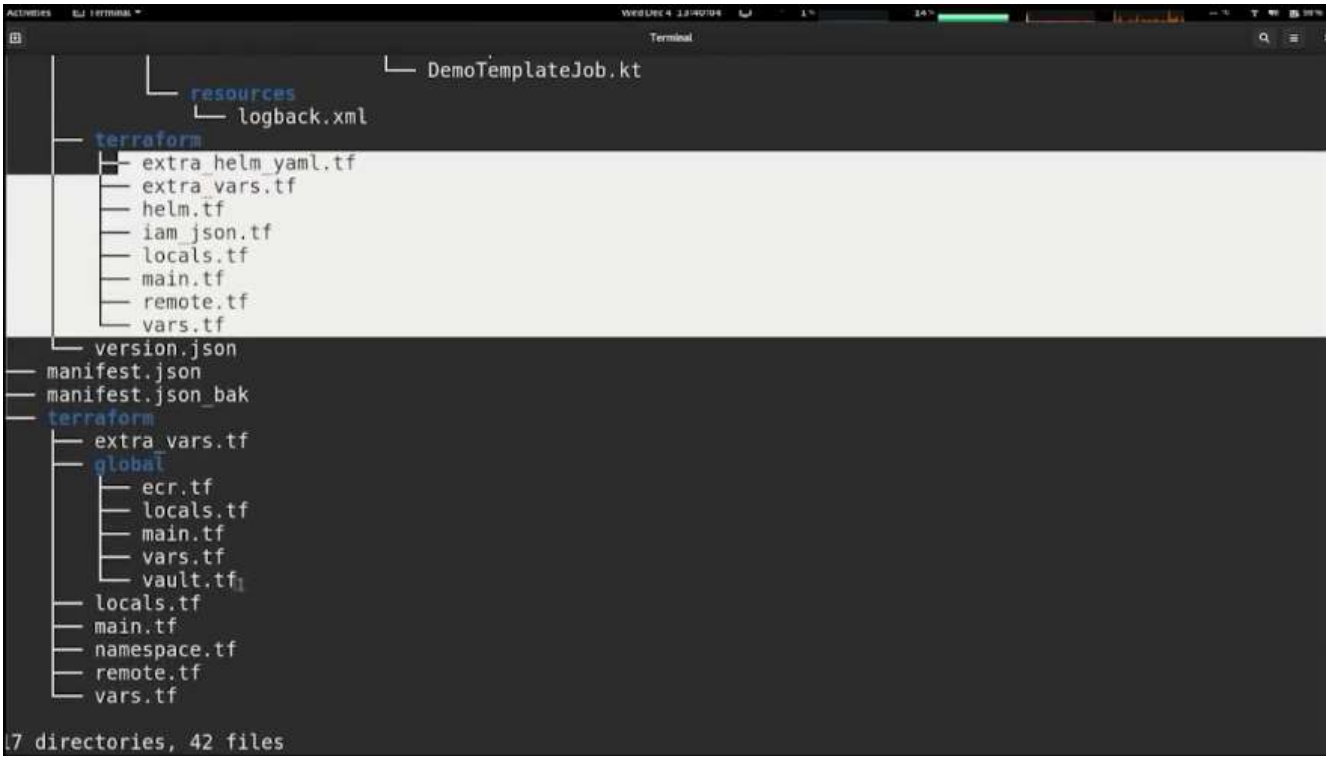


We now have Helm in this directory with the Helm chart


Terminal
Jenkinsfile manifest.json manifest.json_bak terraform
connor:~/git/github.com/birdrides/api/api/service-derived-location
$ tree
helm
service-derived-events
Chart.yaml
values.yaml
Jenkinsfile
location-averager
build.gradle.kts
Dockerfile
extras.groovy
helm
location-averager
Chart.yaml
templates
configmap.yaml
deployment.yaml
_helpers.tpl
ingress.yaml
NOTES.txt
pod_disruption_budget.yml
rbac.yaml
service.yaml
values.yaml
Jenkinsfile
src
main
kotlin


Terminal
logback.xml
terraform
extra_helm_yaml.tf
extra_vars.tf
helm.tf
iam_json.tf
locals.tf
main.tf
remote.tf
vars.tf
version.json
manifest.json
manifest.json_bak
terraform
extra_vars.tf
global
ecr.tf
locals.tf
main.tf
vars.tf
vault.tf
locals.tf
main.tf
namespace.tf
remote.tf
vars.tf
7 directories, 42 files
connor:~/git/github.com/birdrides/api/api/service-derived-location
$


Terminal
values.yaml
Jenkinsfile
location-averager
build.gradle.kts
Dockerfile
extras.groovy
helm
location-averager
Chart.yaml
templates
configmap.yaml
deployment.yaml
_helpers.tpl
ingress.yaml
NOTES.txt
pod_disruption_budget.yml
rbac.yaml
service.yaml
values.yaml
Jenkinsfile
src
main
kotlin
co
bird
flink
demo
DemoSplitFirehoseJob.kt
DemoTemplateJob.kt
resources


Terminal
DemoTemplateJob.kt
resources
logback.xml
terraform
extra_helm_yaml.tf
extra_vars.tf
helm.tf
iam_json.tf
locals.tf
main.tf
remote.tf
vars.tf
version.json
manifest.json
manifest.json_bak
terraform
extra_vars.tf
global
ecr.tf
locals.tf
main.tf
vars.tf
vault.tf
locals.tf
main.tf
namespace.tf
remote.tf
vars.tf
7 directories, 42 files

```
    ├── extra_vars.tf
    ├── helm.tf
    ├── iam_json.tf
    ├── locals.tf
    ├── main.tf
    ├── remote.tf
    └── vars.tf
└── version.json
├── manifest.json
├── manifest.json_bak
└── terraform
    ├── extra_vars.tf
    ├── global
    │   ├── ecr.tf
    │   ├── locals.tf
    │   ├── main.tf
    │   ├── vars.tf
    │   └── vault.tf
    ├── locals.tf
    ├── main.tf
    ├── namespace.tf
    ├── remote.tf
    └── vars.tf

17 directories, 42 files
connor:~/git/github.com/birdrides/api/api/service-derived-location
$ ls
helm  Jenkinsfile  location-averager  manifest.json  manifest.json_bak  terraform
connor:~/git/github.com/birdrides/api/api/service-derived-location
$
```

```
package co.bird.flink.tracks

import ...

private const val DEFAULT_JOB_NAME = "Bird Location Averager Job"

/**
 * Averages locations from Bird tracks over sliding (overlapping) time windows.
 */
class LocationAveragerJob {

  fun build(trackStream: SingleOutputStreamOperator<BirdTrack>): SingleOutputStreamOperator<AveragedBirdLocation> {
    return trackStream
      .filter { it?.location != null && it.location!!.isValidLatLong() }
      .returns(BirdTrack::class.java)
      .keyBy({ it.birdId.toString() }, Types.STRING)
      .window(SlidingProcessingTimeWindows.of(Time.seconds( seconds: 60L), Time.seconds( seconds: 30L)))
      .process()


  }

  companion object {
    @JvmStatic
    fun main(args: Array<String>) {
```

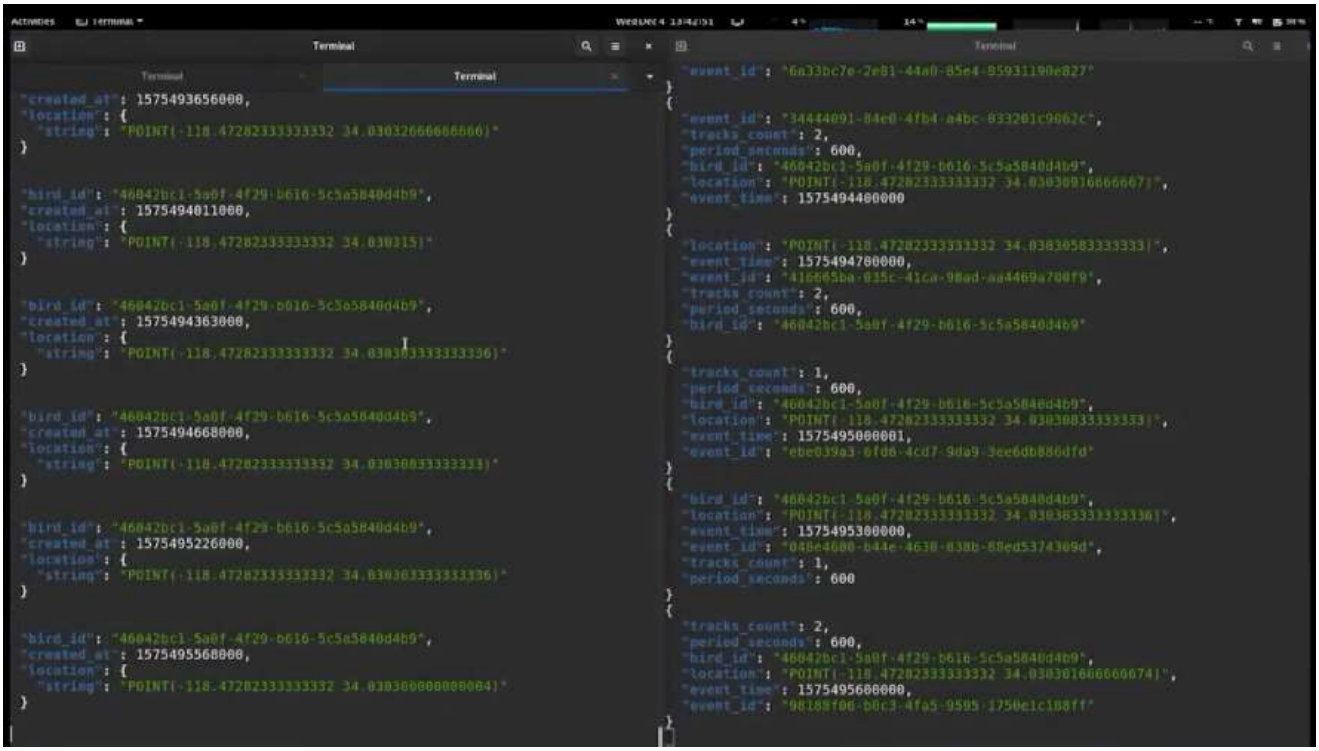
This is our Flink Job code

```
class LocationAveragerJob {

  fun build(trackStream: SingleOutputStreamOperator<BirdTrack>): SingleOutputStreamOperator<AveragedBirdLocation> {
    return trackStream
      .filter { it?.location != null && it.location!!.isValidLatLong() }
      .returns(BirdTrack::class.java)
      function: ProcessWindowFunction<BirdTrack!, AveragedBirdLocation!, String!, TimeWindow!>!
      function: ProcessWindowFunction<BirdTrack!, AveragedBirdLocation!, String!, TimeWindow!>! resultType: TypeInformation<AveragedBirdLocation!>!
                                                                                                  seconds( seconds: 30L)))
      .process(AverageLocationFunction())
```

```
onnor:~
$ kubectl get pods -n service-derived-locations
AME                                          READY   STATUS    RESTARTS   AGE
link-location-averager-jobmanager             1/1     Running   0          12h
link-location-averager-taskmanager-79746fdbdc-kfwm7   1/1     Running   0          12h
link-location-averager-taskmanager-79746fdbdc-tk222   1/1     Running   0          12h
onnor:~
$ kubectl get pods -n service-derived-locations
AME                                          READY   STATUS    RESTARTS   AGE
link-location-averager-jobmanager             1/1     Running   0          14h
link-location-averager-taskmanager-79746fdbdc-kfwm7   1/1     Running   0          14h
link-location-averager-taskmanager-79746fdbdc-tk222   1/1     Running   0          14h
onnor:~
$
```

```
r Code: 404 Not Found; Request ID: DF83B4A3BB8092C5; S3 Extended Request ID: F9dLHIPrpSlNNOVwuA1DWXEOkG5ksREiuKo5GOlEP3aFGlNrOgmUu6Pby4
uo5nOKKnoNOtyHok=), S3 Extended Request ID: F9dLHIPrpSlNNOVwuA1DWXEOkG5ksREiuKo5GOlEP3aFGlNrOgmUu6Pby4ouo5nOKKnoNOtyHok=], RequestType=
GetObjectMetadataRequest], AWSRequestID=[DF83B4A3BB8092C5], HttpClientPoolPendingCount=0, RetryCapacityConsumed=0, HttpClientPoolAvail
leCount=0, RequestCount=1, Exception=1, HttpClientPoolLeasedCount=0, ClientExecuteTime=[26.871], HttpClientSendRequestTime=[0.049], Ht
RequestTime=[26.416], RequestSigningTime=[0.148], CredentialsRequestTime=[0.003, 0.0], HttpClientReceiveResponseTime=[16.138],
C
onnor:~
$ kubectl logs -f -n service-derived-locations flink-location-averager-taskmanager-79746fdbdc-kfwm7
```



We are reading a stream of events coming out of Kafka on the left, then we strip out the tracks for an individual Bird