# How We Run 100 Airflow Environments and Millions of Tasks as a Part Time job Using Kubernetes

Michael Juster

BALYASNY
ASSET
MANAGEMENT

# Airflow and Kubernetes at BAM

Michael Juster, Senior Platform Engineer
September 2024

# About BAM

**Balyasny Asset Management (BAM) is a diversified global investment firm founded in 2001 with over $20 billion in assets under management.**

**We have more than 100 teams who run a variety of strategies that benefit from orchestration and parallelization.**

**We're Hiring!**
**https://bambusdev.my.site.com/s/**

# About Me

Platform Engineer at BAM for 5+ Years.  Designed our system with my colleagues

Software Engineer at Groupon.  Designed their Airflow System

Started my career as a Trader

I have three kids who all think that anyone who is on YouTube is famous!

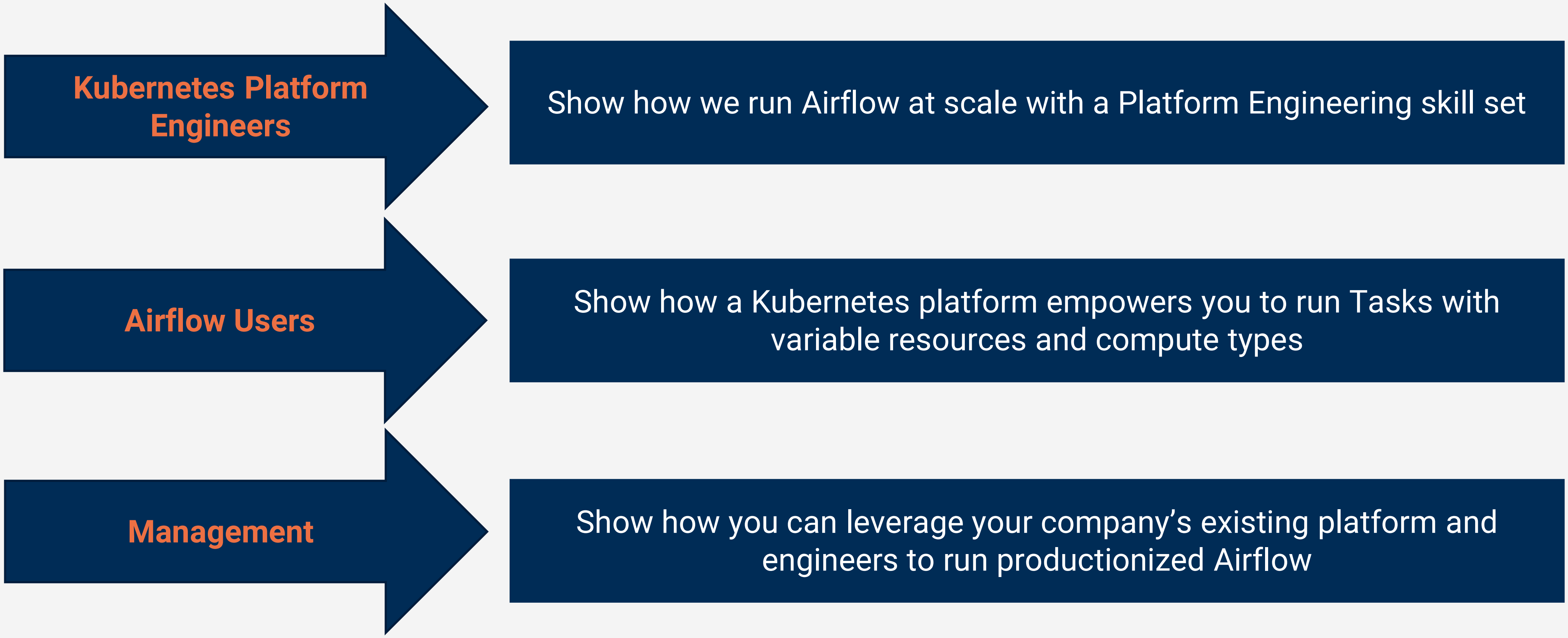
# Airflow At BAM

26,000 concurrent CPUs and 150 TB of concurrent RAM Usage at peak times, and Airflow Tasks are a big part of that

Scheduled Tasks are part of the Workloads of every team

Airflow is not just a “Data” tool.  It is a Scheduling tool.

# Goals Of This Presentation



**Kubernetes Platform Engineers**

Show how we run Airflow at scale with a Platform Engineering skill set

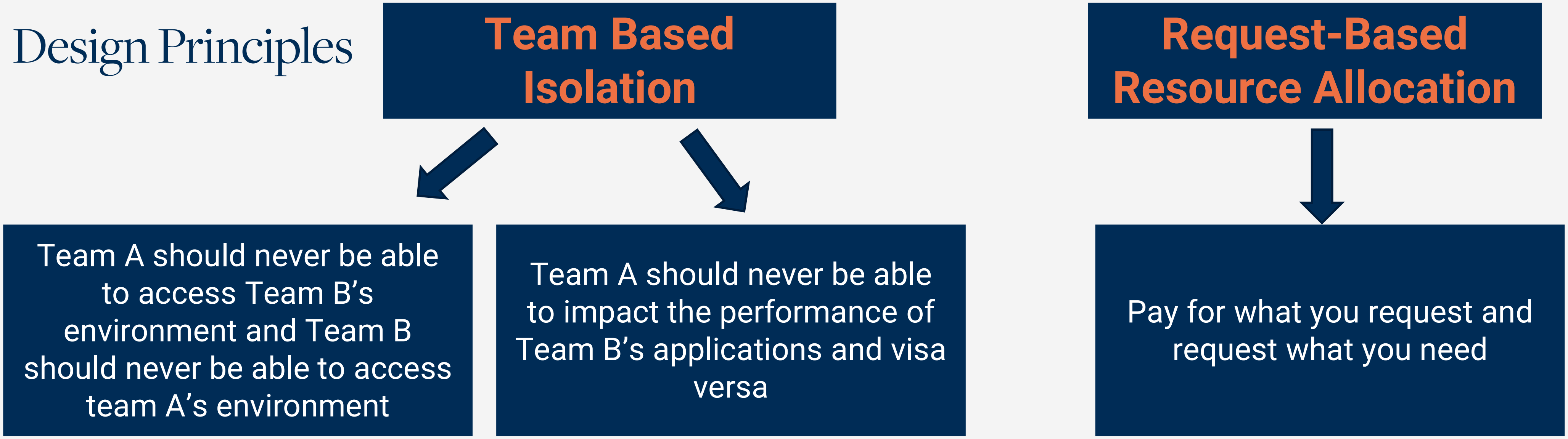
**Airflow Users**

Show how a Kubernetes platform empowers you to run Tasks with variable resources and compute types

**Management**

Show how you can leverage your company’s existing platform and engineers to run productionized Airflow

# Design Principles



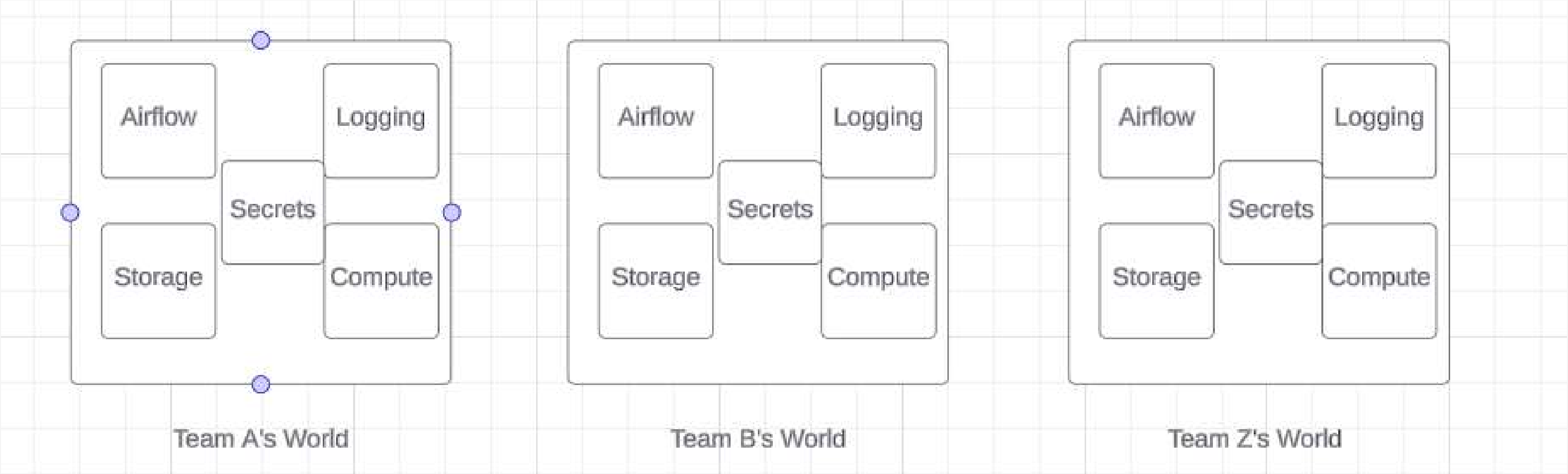
## Team Based Isolation

Team A should never be able to access Team B's environment and Team B should never be able to access team A's environment

Team A should never be able to impact the performance of Team B's applications and visa versa

## Request-Based Resource Allocation

Pay for what you request and request what you need



For illustrative purposes only.

# Kubernetes At A High Level

Kubernetes is extremely good at running containers.

Everything in our platform is containerized whether Airflow Tasks or the environment itself.

# Container Extreme Basics

Containers are created from Images

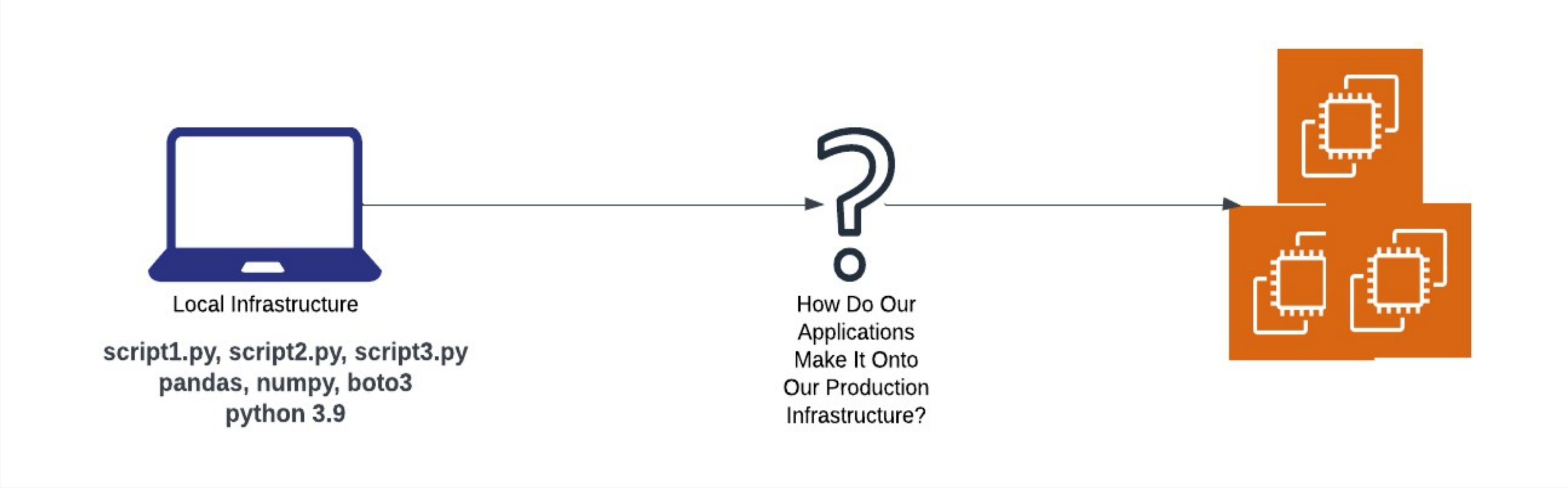
Images are fancy zip files with an application’s code and its dependencies

Our Users are responsible for their images

- The fancy zip files that house the scripts run by their Airflow Tasks

# You Need An Artifact

No matter what you need to solve the problem of how your applications and their dependencies make it onto and run on production infrastructure



Local Infrastructure

script1.py, script2.py, script3.py
pandas, numpy, boto3
python 3.9

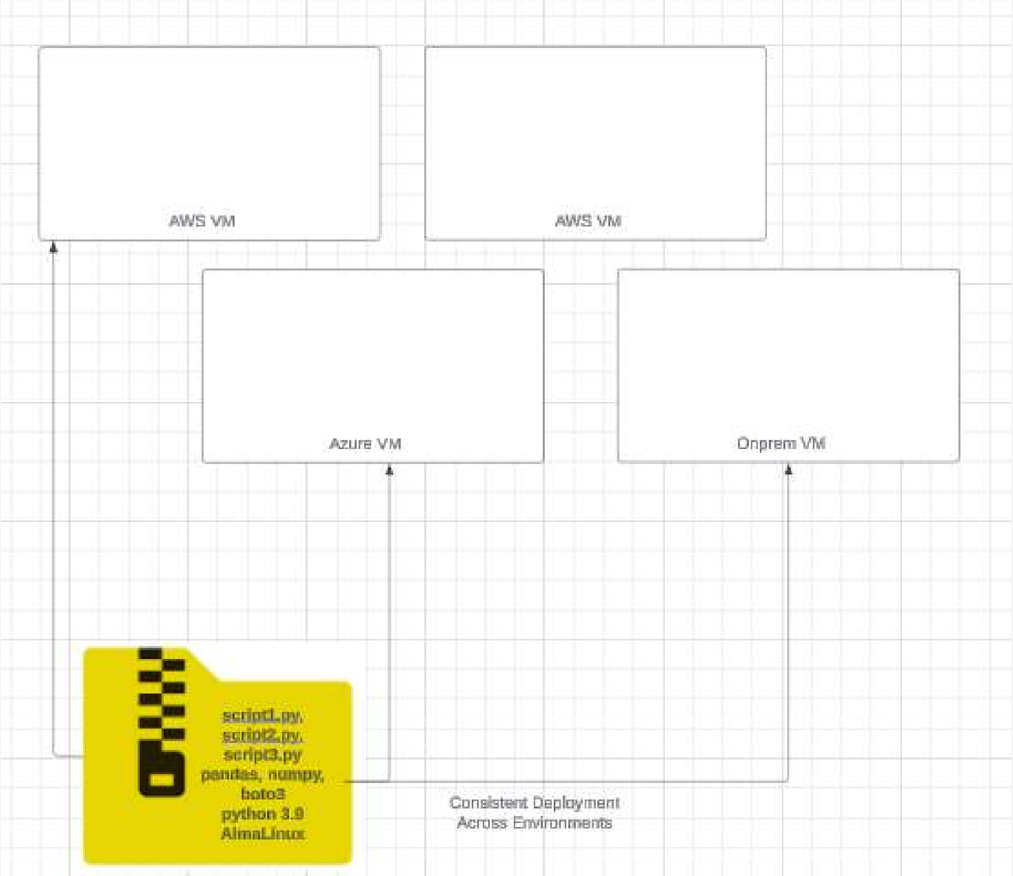
How Do Our Applications Make It Onto Our Production Infrastructure?

For illustrative purposes only.

# Why We Like Images

Reliable Code Packaging

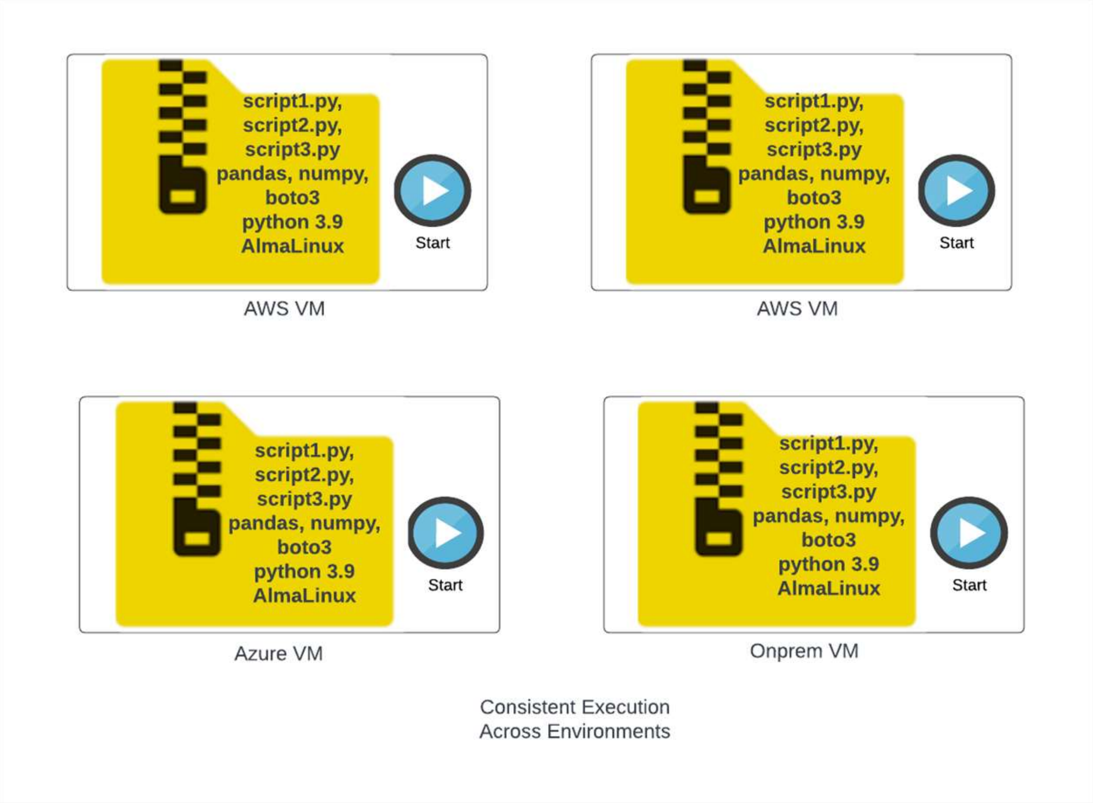
Simplifies Deployment: Images streamline the process of distributing code across different environments



For illustrative purposes only.

# Running Images As Containers

**Start Images As Containers**



For illustrative purposes only.

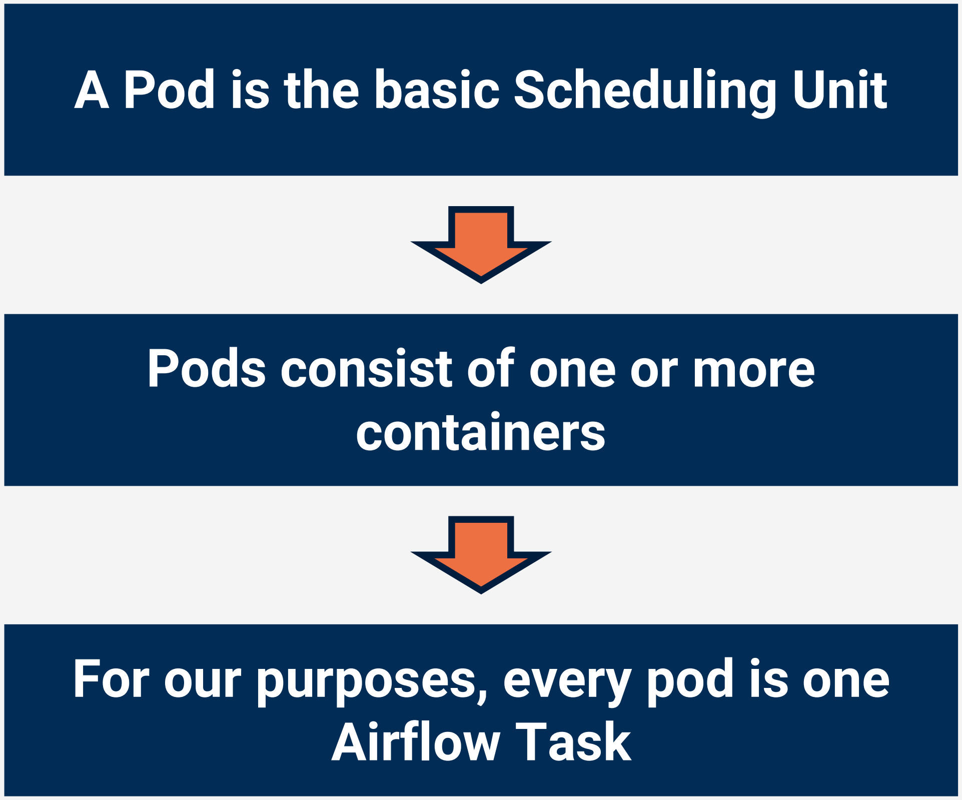
# Recap

Images are fancy zip files

They are a great way to package the code run by Airflow Tasks

Platform Engineers understand them very well

# A Bit More About Kubernetes

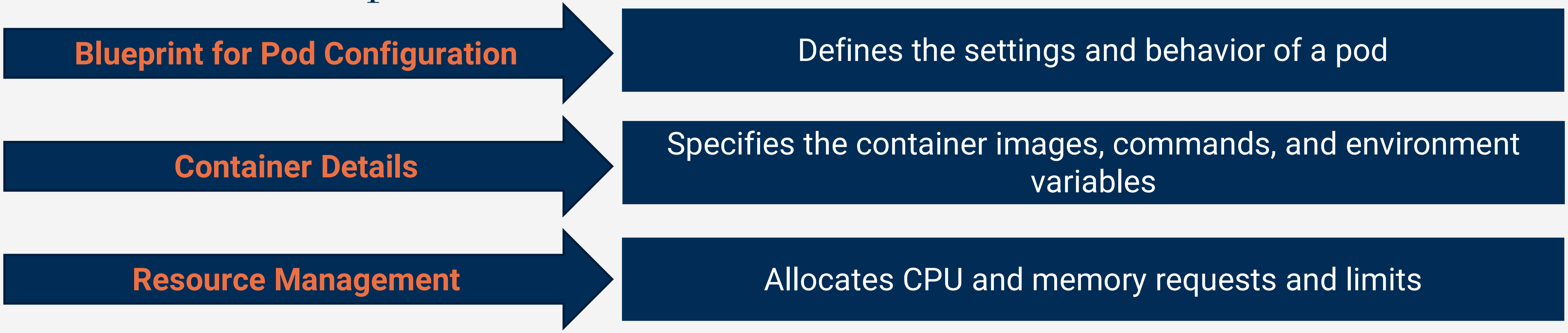


A Pod is the basic Scheduling Unit

Pods consist of one or more containers

For our purposes, every pod is one Airflow Task

# Kubernetes Pod Specs



**Blueprint for Pod Configuration**

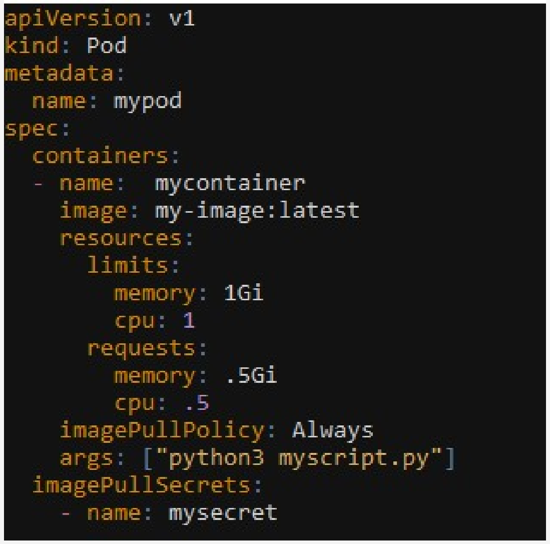
Defines the settings and behavior of a pod

**Container Details**

Specifies the container images, commands, and environment variables

**Resource Management**

Allocates CPU and memory requests and limits



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name:  mycontainer
    image: my-image:latest
    resources:
      limits:
        memory: 1Gi
        cpu: 1
      requests:
        memory: .5Gi
        cpu:  .5
    imagePullPolicy: Always
    args: ["python3 myscript.py"]
  imagePullSecrets:
    - name: mysecret
```

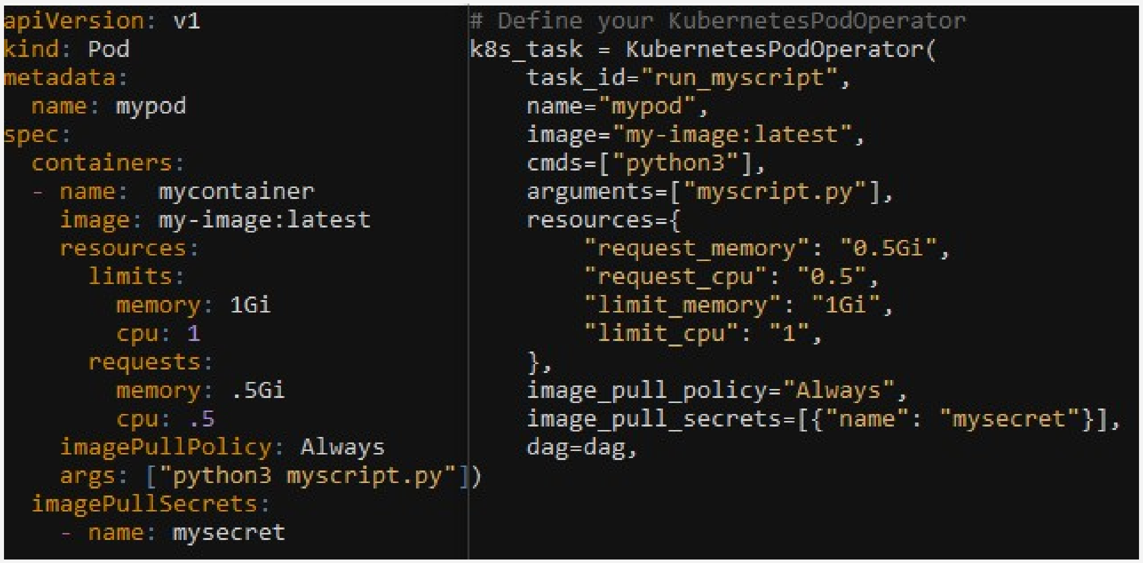
For illustrative purposes only.

# A Task Is Just Another Pod

Platform Engineers understand Pod Specs very well!

They understand the idea that Airflow is just submitting a bunch of Pods

They can help users with adjustments even if they don't understand Operators

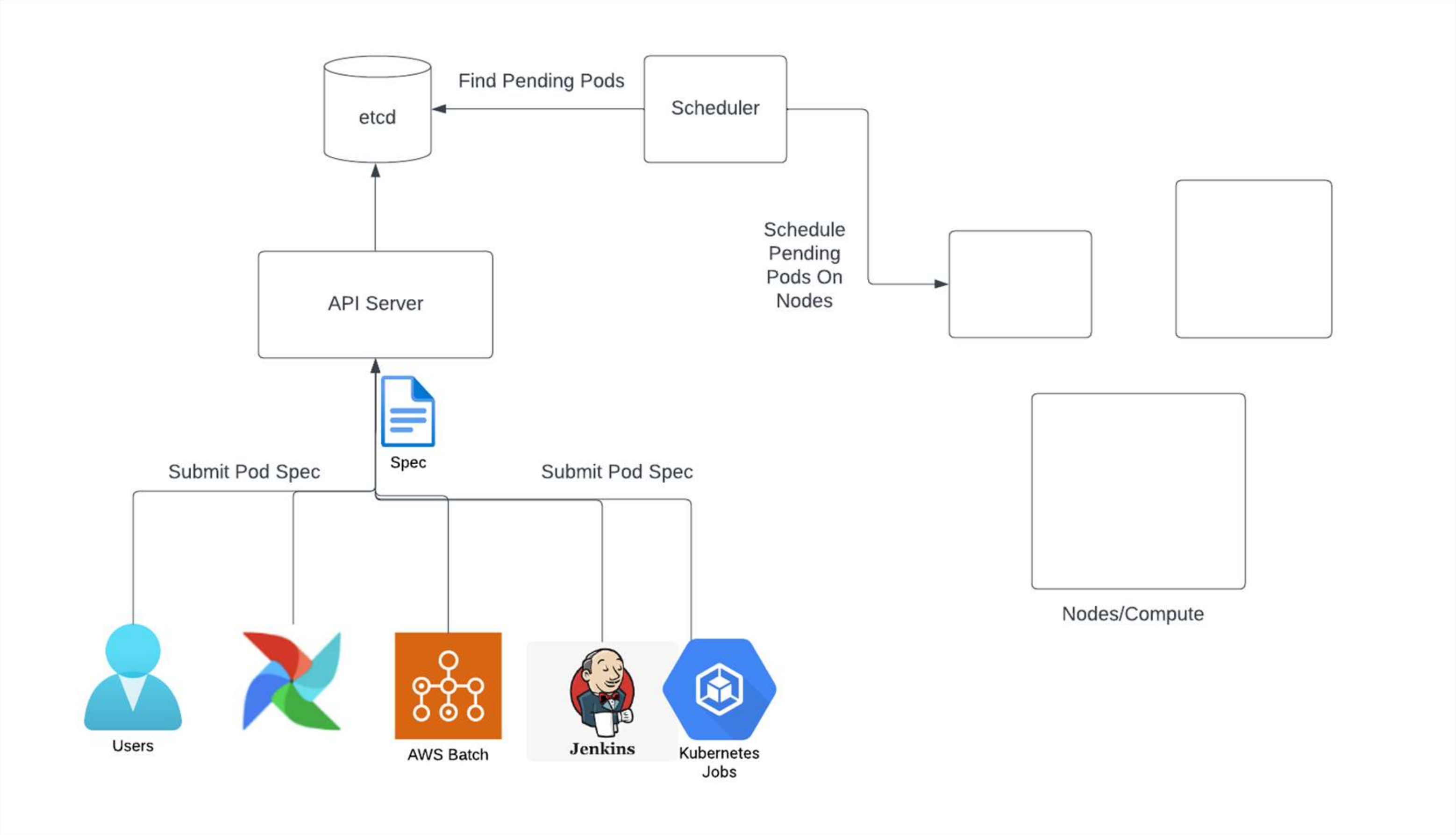


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name:  mycontainer
    image: my-image:latest
    resources:
      limits:
        memory: 1Gi
        cpu: 1
      requests:
        memory: .5Gi
        cpu: .5
    imagePullPolicy: Always
    args: ["python3 myscript.py"]
  imagePullSecrets:
    - name: mysecret
```

```python
# Define your KubernetesPodOperator
k8s_task = KubernetesPodOperator(
    task_id="run_myscript",
    name="mypod",
    image="my-image:latest",
    cmds=["python3"],
    arguments=["myscript.py"],
    resources={
        "request_memory": "0.5Gi",
        "request_cpu": "0.5",
        "limit_memory": "1Gi",
        "limit_cpu": "1",
    },
    image_pull_policy="Always",
    image_pull_secrets=[{"name": "mysecret"}],
    dag=dag,
)
```

For illustrative purposes only.

# It’s All The Same Game

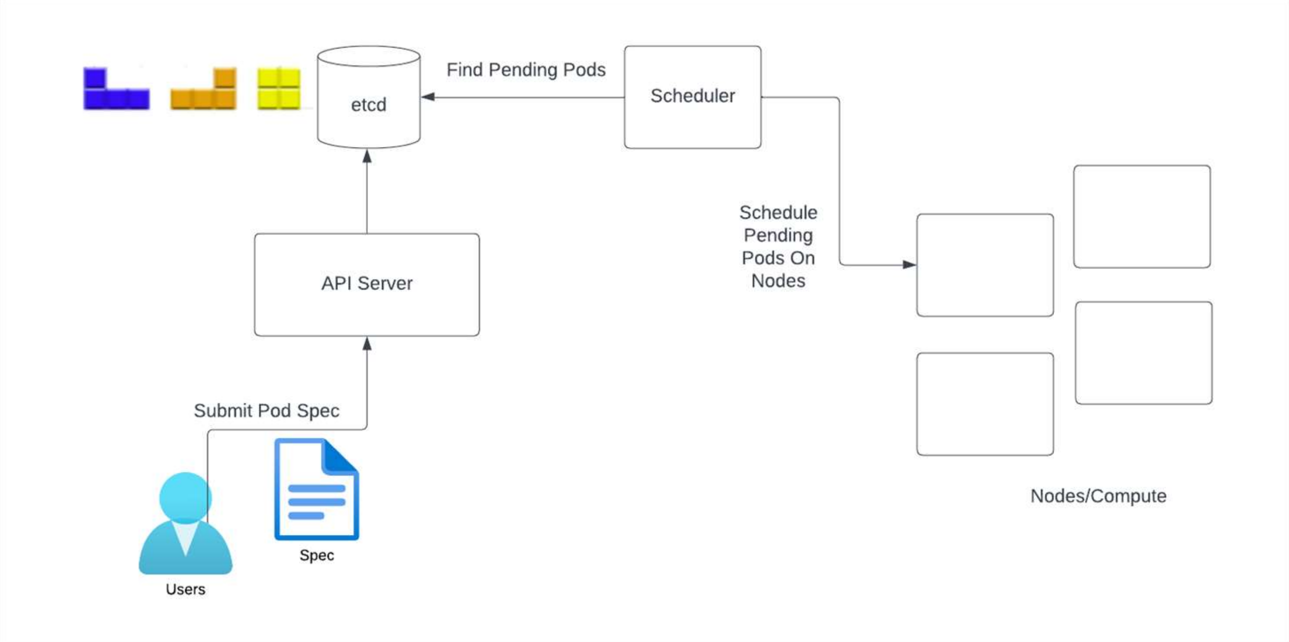


For illustrative purposes only.

# Understanding The Container Scheduling Game

A Continuous game of Tetris

Users request resources and select compute type and Kubernetes does the rest

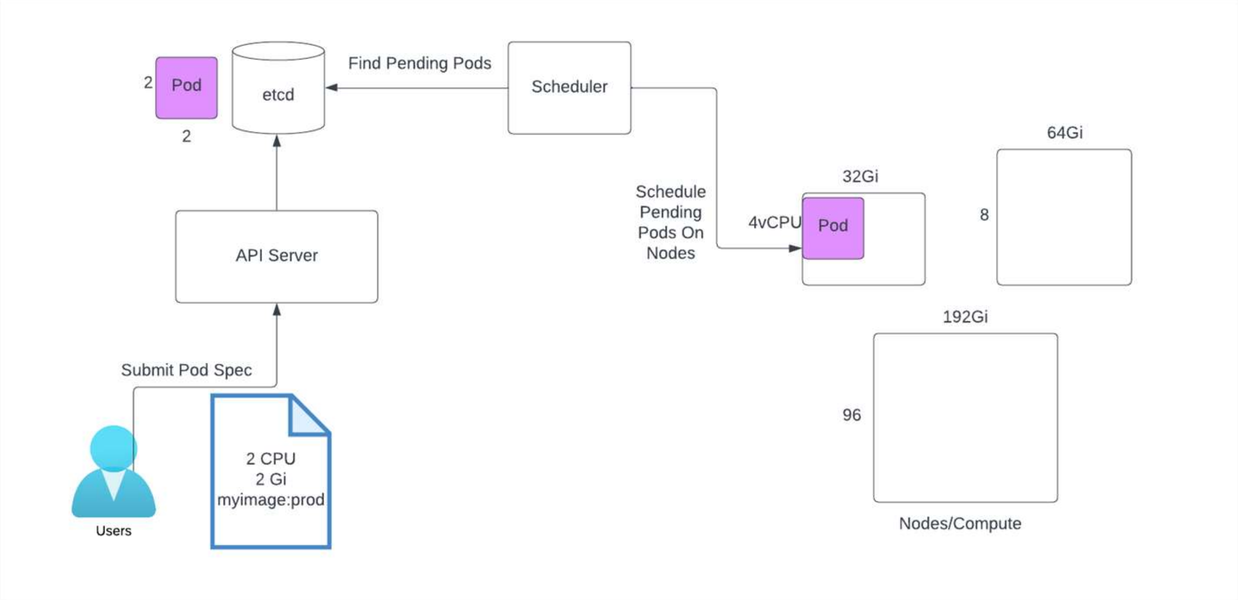


For illustrative purposes only.

# The Game Of Tetris

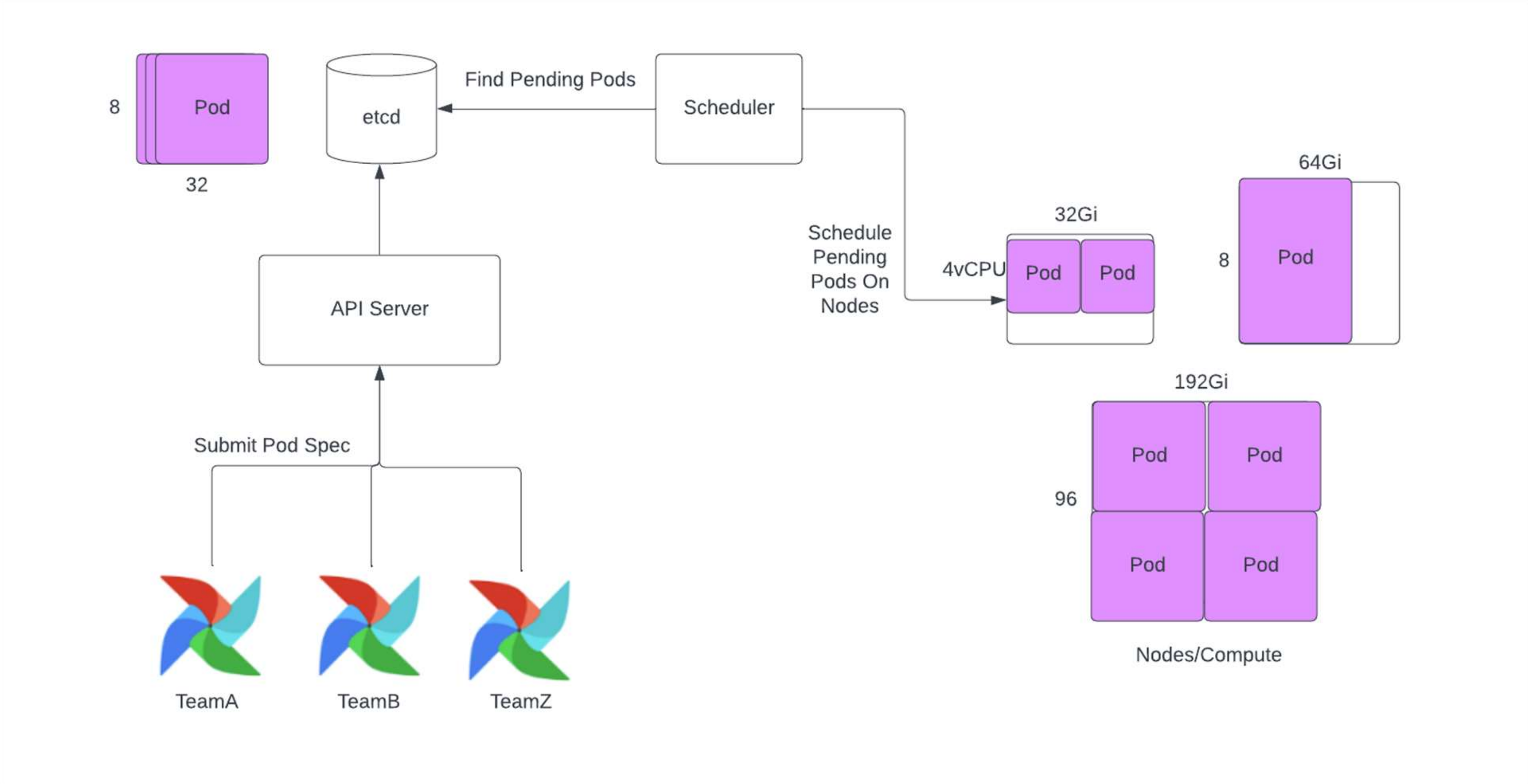
Imagine trying to play Tetris if you did not know the shape of the Tetris block before placing it?

Pods and Nodes/VMs have dimensions



For illustrative purposes only.

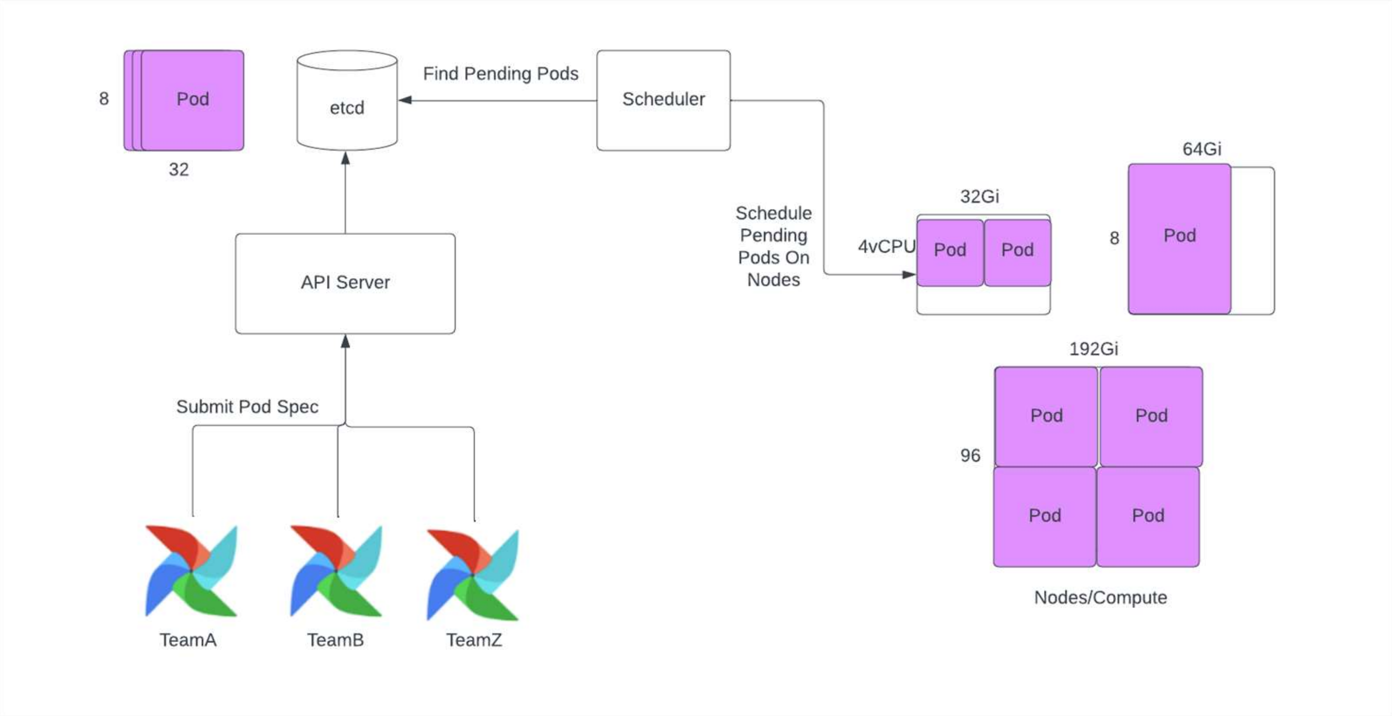
# The Game Of Tetris Continued



For illustrative purposes only.

# Dynamic Scaling

## Our Airflow Environments Continue to submit Tasks but there's no more Capacity.  Now what happens?

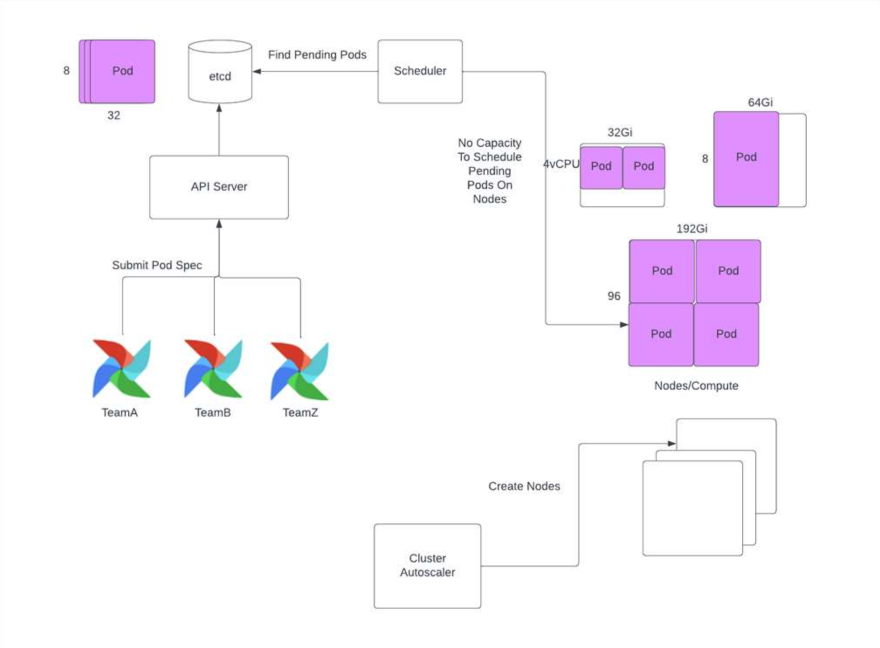


For illustrative purposes only.

# Cluster Autoscaler

Automatically adjusts the number of nodes in a cluster

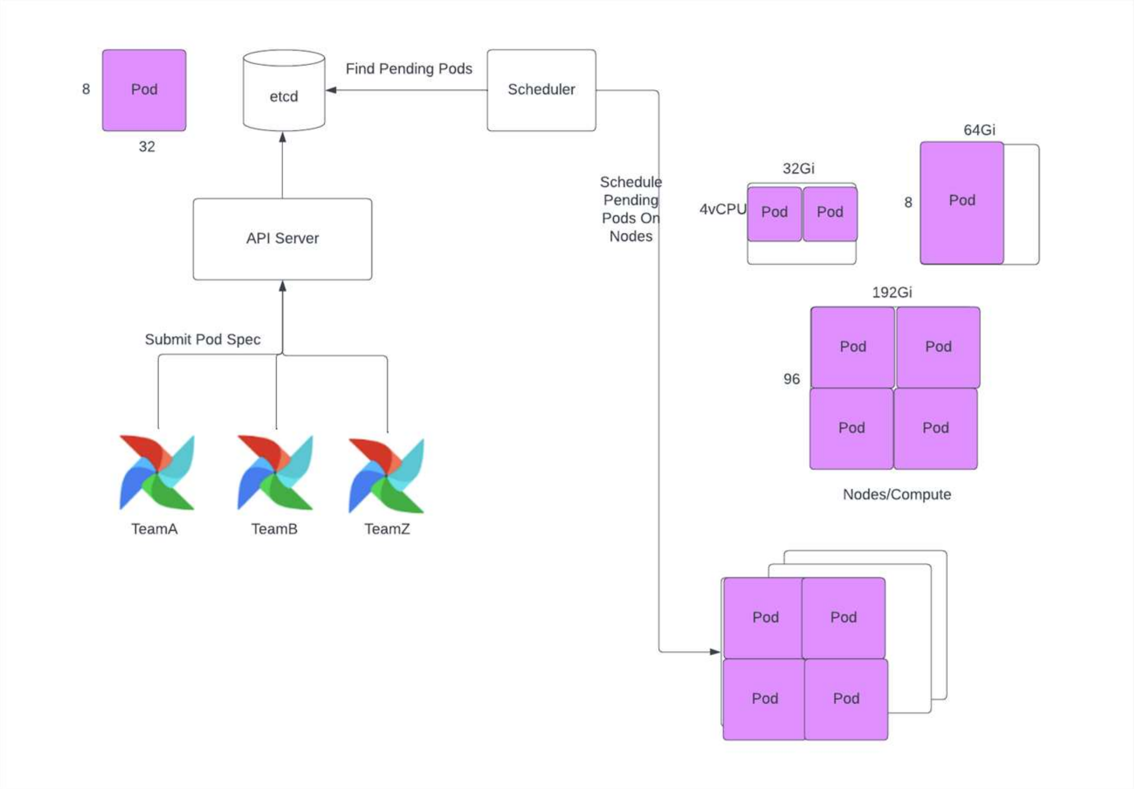
Scales up when there are pending pods that cannot be scheduled due to insufficient resources



For illustrative purposes only.

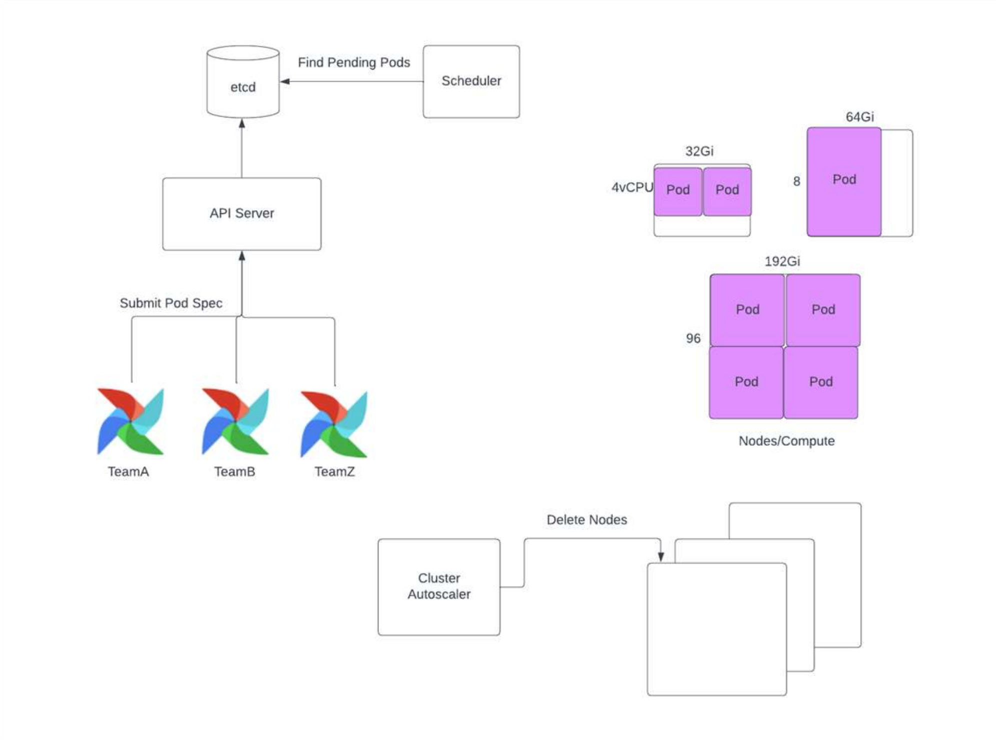
© Balyasny Asset Management, L.P.

# Cluster Autoscaler Efficient Scaling



# Cluster Autoscaler Efficient Scaling Continued

## Scales down when nodes are underutilized, ensuring efficient resource use

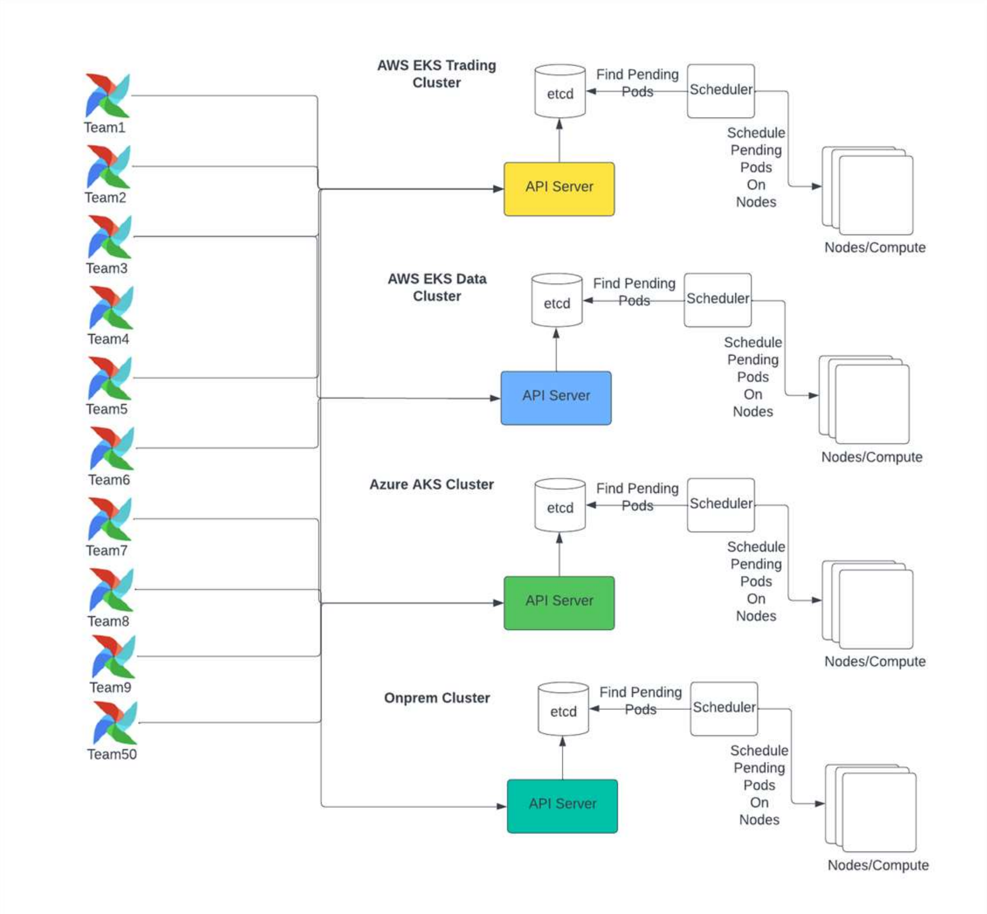


For illustrative purposes only.

# Big Picture Scale

Teams can submit their pod specifications to any Kubernetes cluster, all integrated with the Cluster Autoscaler

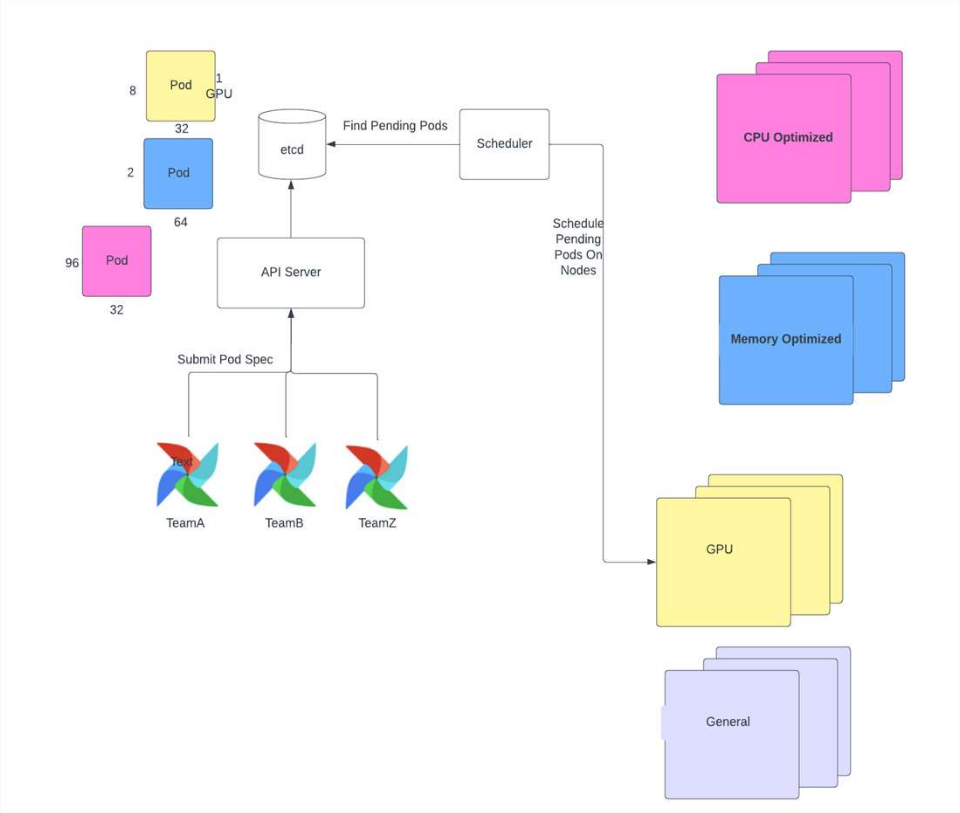
Consistent user experience across clusters



# Teams Select Compute They Need

Teams choose from a range of compute options depending on their Task requirements

CPU Optimized, Memory Optimized, GPU, and General



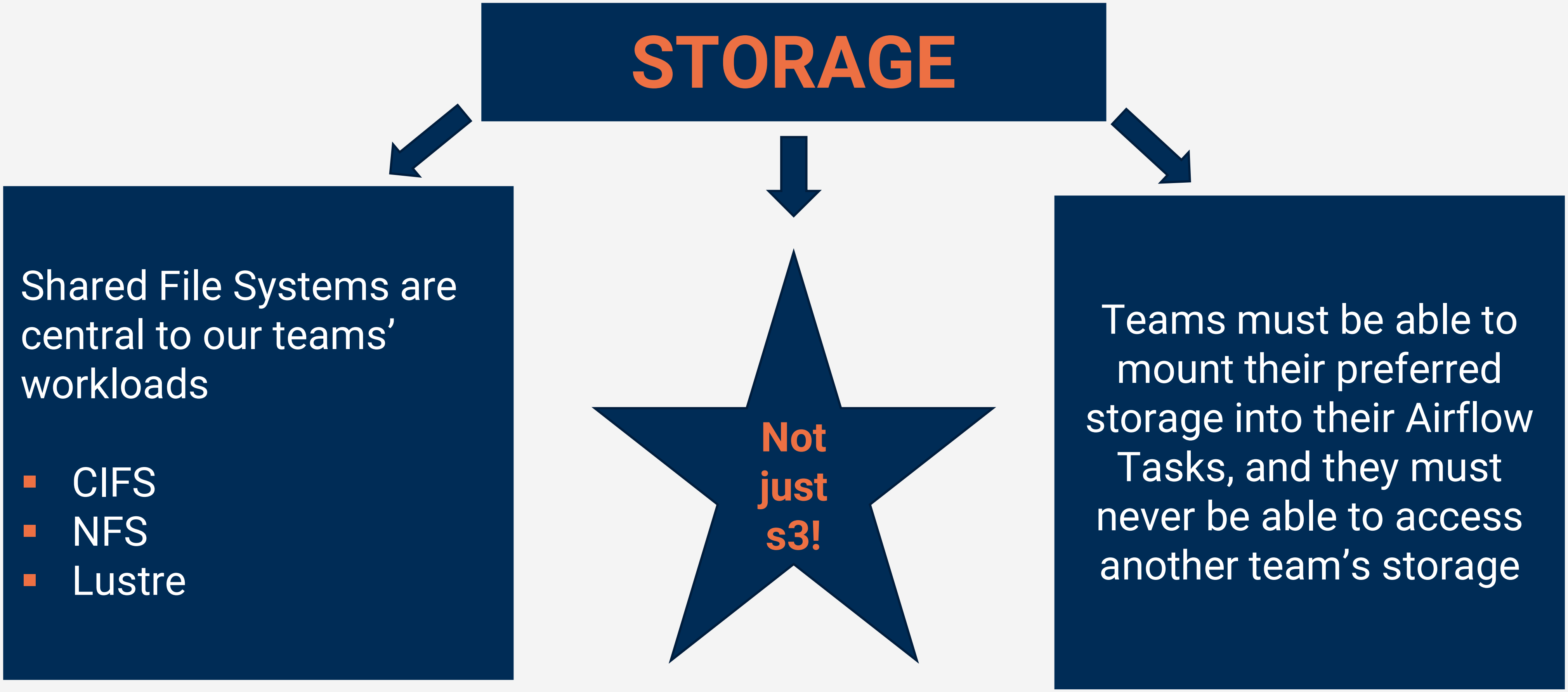
For illustrative purposes only.

# Recap

The Kubernetes Pod Operator runs every Task as a Pod

To a Platform Engineer there is no difference between an Airflow Task's Pod and any other Pod

The existing Kubernetes Platform enables scalability that perfectly suits job based workloads



# STORAGE

Shared File Systems are central to our teams' workloads

- CIFS
- NFS
- Lustre

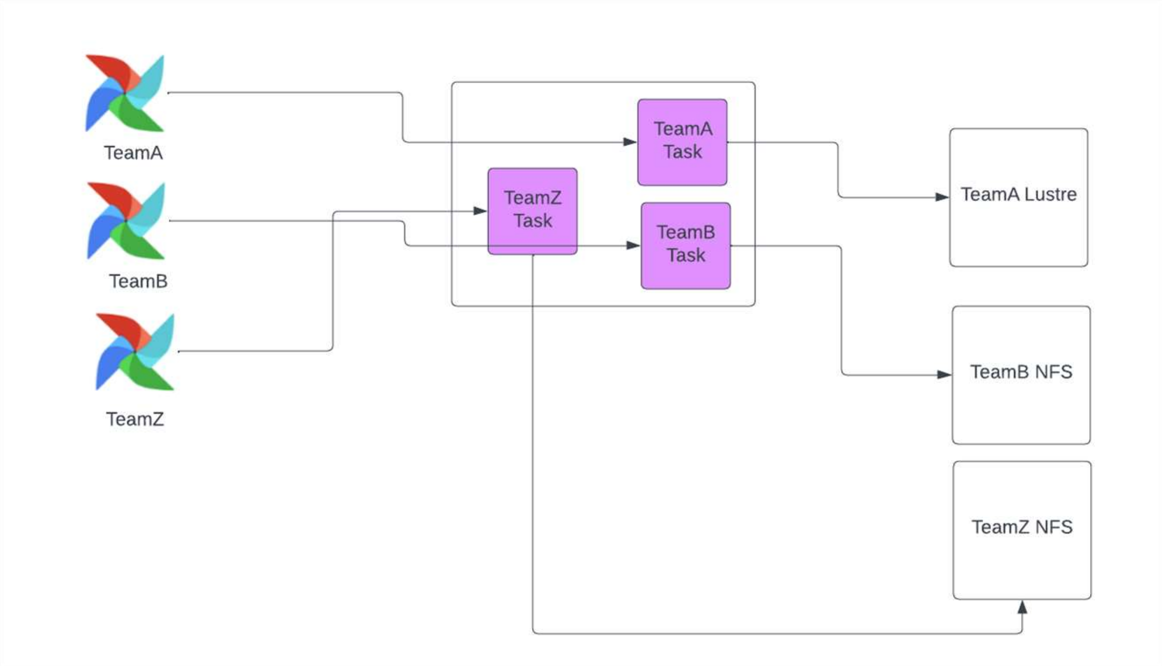
Not just s3!

Teams must be able to mount their preferred storage into their Airflow Tasks, and they must never be able to access another team's storage

# Mounting Storage In Tasks

Users request storage via Kubernetes Persistent Volume Claims (PVCs)

Persistent Volume Claims (PVCs) enable users to request and use storage without needing to understand the underlying storage details.

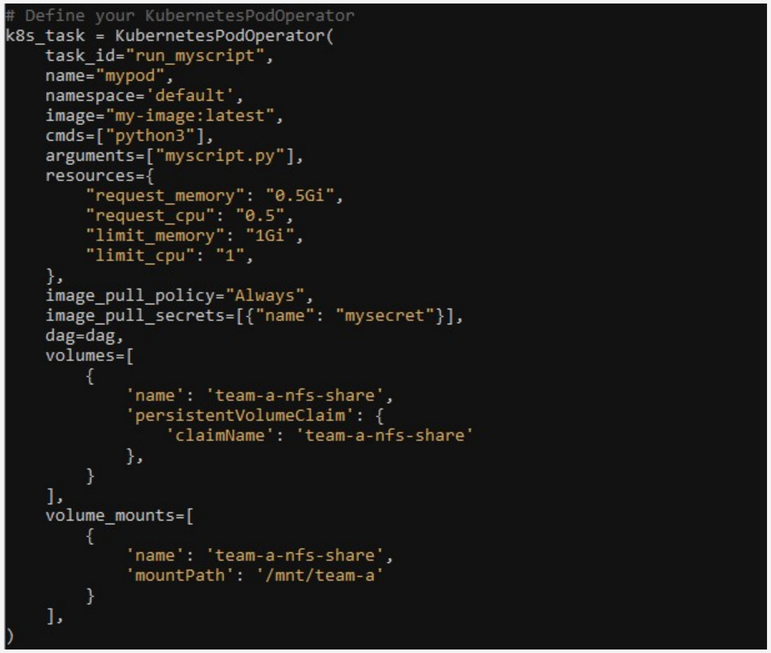


For illustrative purposes only.

# Mounting Storage In Tasks Continued

Teams only need to understand how to reference their Persistent Volume Claims in their Kubernetes Pod Operator calls

Platform Engineers and the platform itself handle the rest



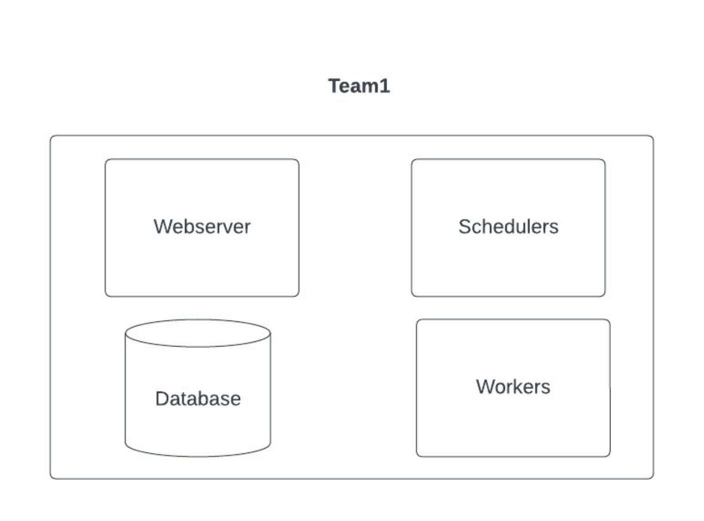
```
# Define your KubernetesPodOperator
k8s_task = KubernetesPodOperator(
    task_id="run_myscript",
    name="mypod",
    namespace='default',
    image="my-image:latest",
    cmds=["python3"],
    arguments=["myscript.py"],
    resources={
        "request_memory": "0.5Gi",
        "request_cpu": "0.5",
        "limit_memory": "1Gi",
        "limit_cpu": "1",
    },
    image_pull_policy="Always",
    image_pull_secrets=[{"name": "mysecret"}],
    dag=dag,
    volumes=[
        {
            'name': 'team-a-nfs-share',
            'persistentVolumeClaim': {
                'claimName': 'team-a-nfs-share'
            },
        }
    ],
    volume_mounts=[
        {
            'name': 'team-a-nfs-share',
            'mountPath': '/mnt/team-a'
        }
    ],
)
```

For illustrative purposes only.

# Deploying Team Based Airflow Environments

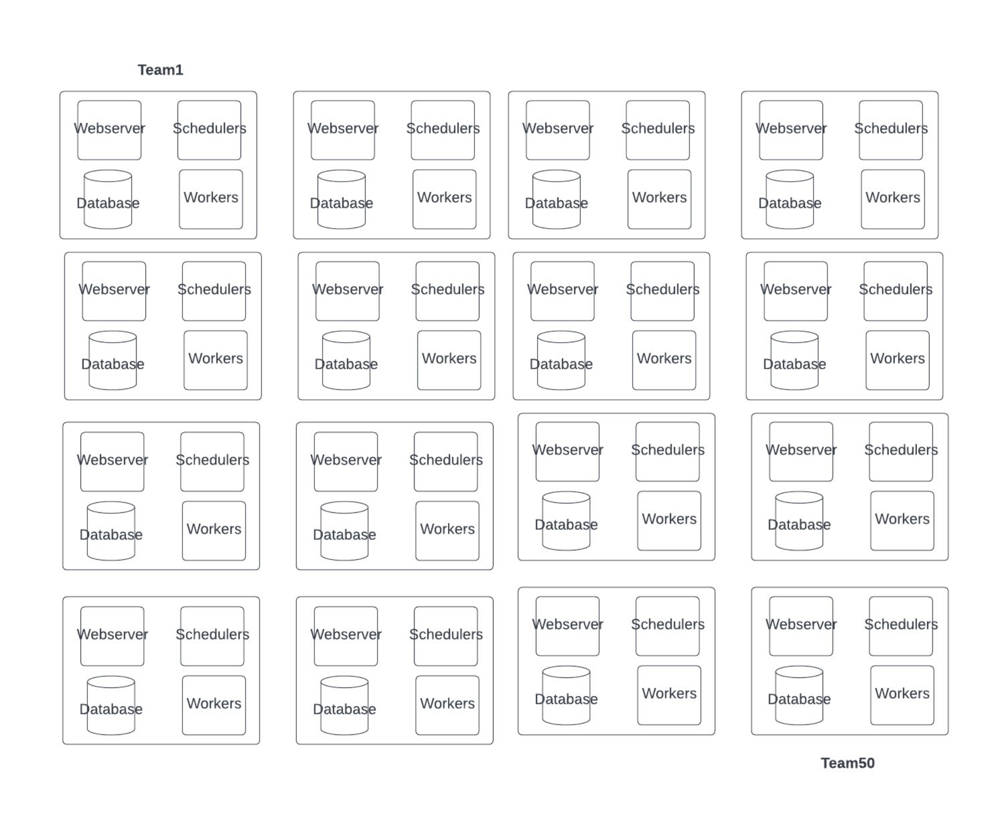
We use a variation of the community helm chart to deploy team-based Airflow Environments

Every team gets their own Webserver, Schedulers, Database, and Workers



For illustrative purposes only.

# Team Based Airflow Environments Big Picture



Team1

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

Webserver Schedulers Database Workers

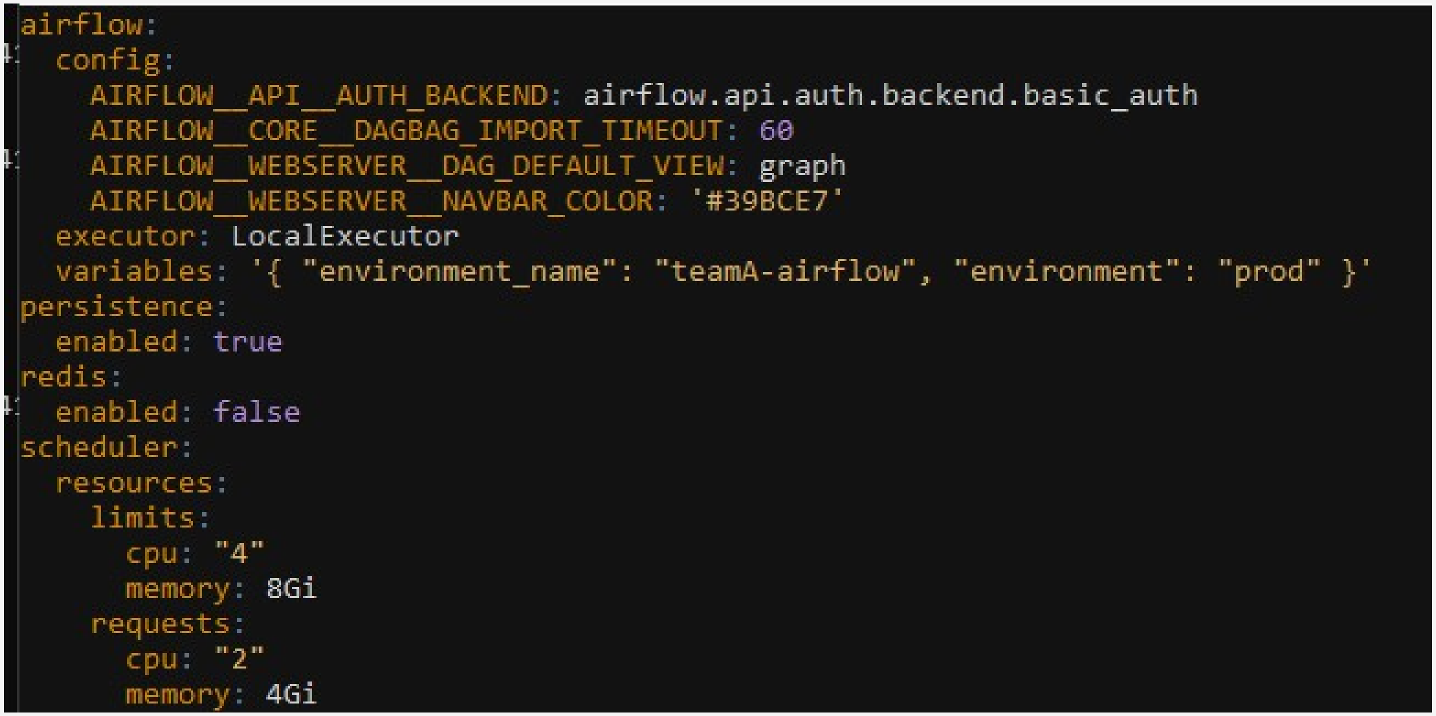
Webserver Schedulers Database Workers

Team50

For illustrative purposes only.

# Easy To Install And Setup Environments

1. Fill out a Values File
2. Open PR
3. Merge PR
4. ????????
5. Profit

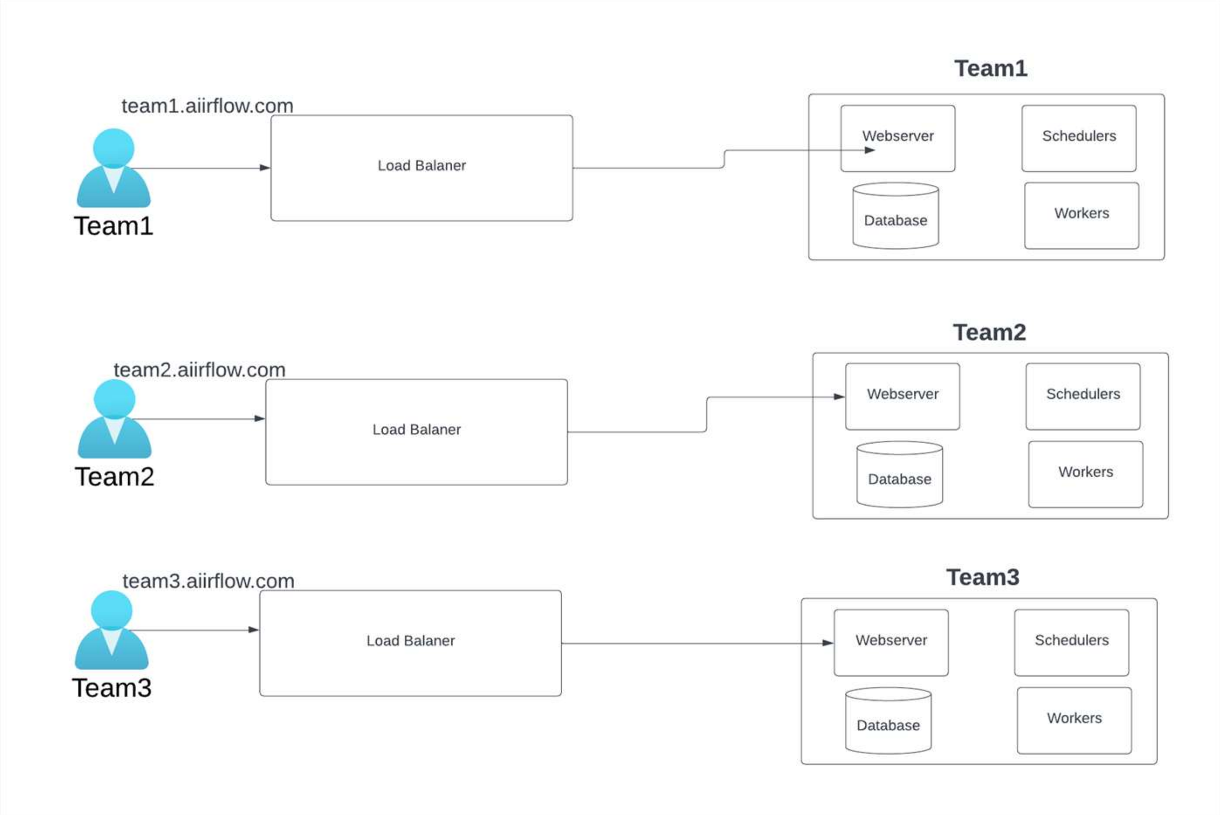


```yaml
airflow:
  config:
    AIRFLOW__API__AUTH_BACKEND: airflow.api.auth.backend.basic_auth
    AIRFLOW__CORE__DAGBAG_IMPORT_TIMEOUT: 60
    AIRFLOW__WEBSERVER__DAG_DEFAULT_VIEW: graph
    AIRFLOW__WEBSERVER__NAVBAR_COLOR: '#39BCE7'
  executor: LocalExecutor
  variables: '{ "environment_name": "teamA-airflow", "environment": "prod" }'
persistence:
  enabled: true
redis:
  enabled: false
scheduler:
  resources:
    limits:
      cpu: "4"
      memory: 8Gi
    requests:
      cpu: "2"
      memory: 4Gi
```

For illustrative purposes only.

# What Else An Environment Comes With

## Each team's environment comes with a Load Balancer, DNS Entry, Logging, Metrics and Alerting



For illustrative purposes only.

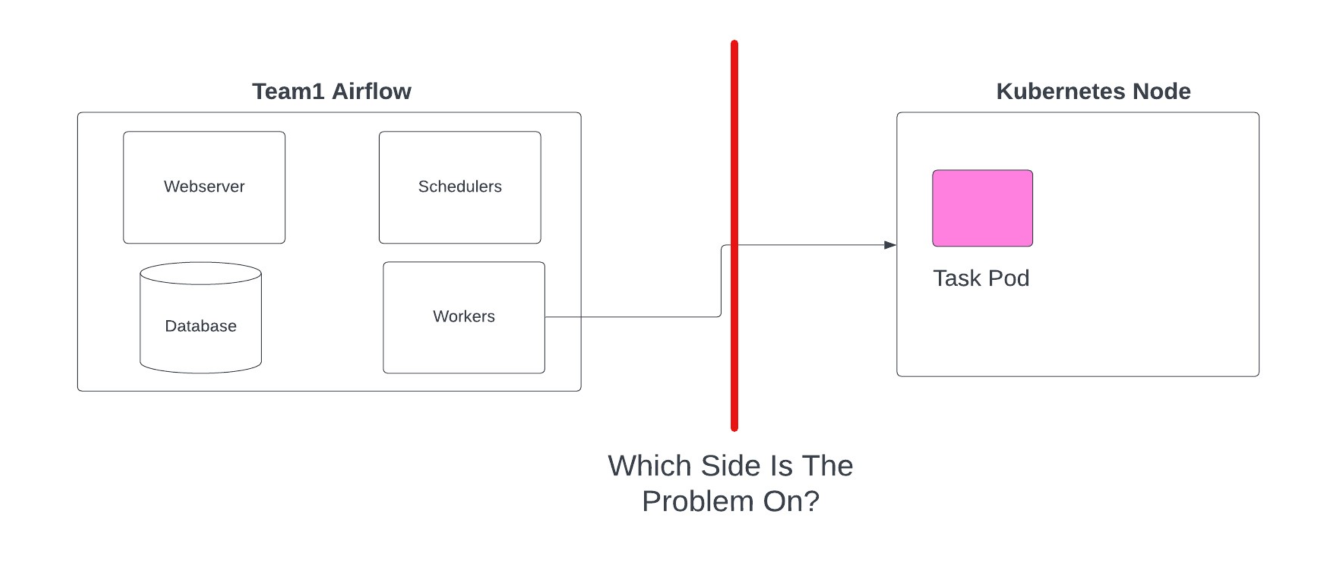
# Troubleshooting

Why didn’t my Task Start?

Why did my Task take “X” minutes to start?

Why did my Task Fail?

Why is my task running slowly compared to before?



# Challenges

**Consistent Upgrades**

Ensuring version consistency across multiple teams

**Keeping Up With Releases**

Staying current with frequent Airflow updates and promptly making them available to all users requires significant effort and coordination.

**Expertise Requirement**

A few members of the overall team need to understand Airflow well, which can interfere with hiring decisions

**Critical Task Reliability**

Task failures are unacceptable and can be difficult to diagnose, posing challenges to maintain workflow stability.

# Where We're Headed

Hybrid of Vendor Managed Airflow Environments and BAM Managed

Batch Scheduling

Systematic

B.A.M. BALYASNY ASSET MANAGEMENT

# THANKS FOR LISTENING!!

This information is furnished on a confidential basis. You agree not to reproduce, redistribute or disclose this material, in whole or in part, to any other person. The information provided in this document are as of the date stated herein and are subject to change without prior notice or any notice whatsoever. This information is not intended to be investment advice and should not be used as such. Any graphs, charts, illustrations and/or formulas are for illustrative purposes only and cannot be used for investment decision making purposes and are not being included to assist in these decisions. We believe the information is accurate as of the date of this report, without error and from reliable sources; however, BAM makes no warranty or representation with respect to the accuracy, validity or completeness of the information.

# Questions?

Michael Juster