

One Controller to Rule Them All - Taming Multiple Orchestrators at Uber Scale

- Deployment
- Instance health
- Environment history
- Capacity
- Continuous Profiler
- Incident mitigation
- Networking
- Settings

Other resources

- [Logs](#)
- [Healthline](#)
- [Statistics](#)
- [Service Metrics](#)

Deployment

Deploy

Hotfix

Restart

Current revisions [Show details](#)

027e57c Prod. version of CD micro repo test service

Target revision

DCA

Instances dca22: 2

dca22 > dca22-kubernetes-stateless01



2 running

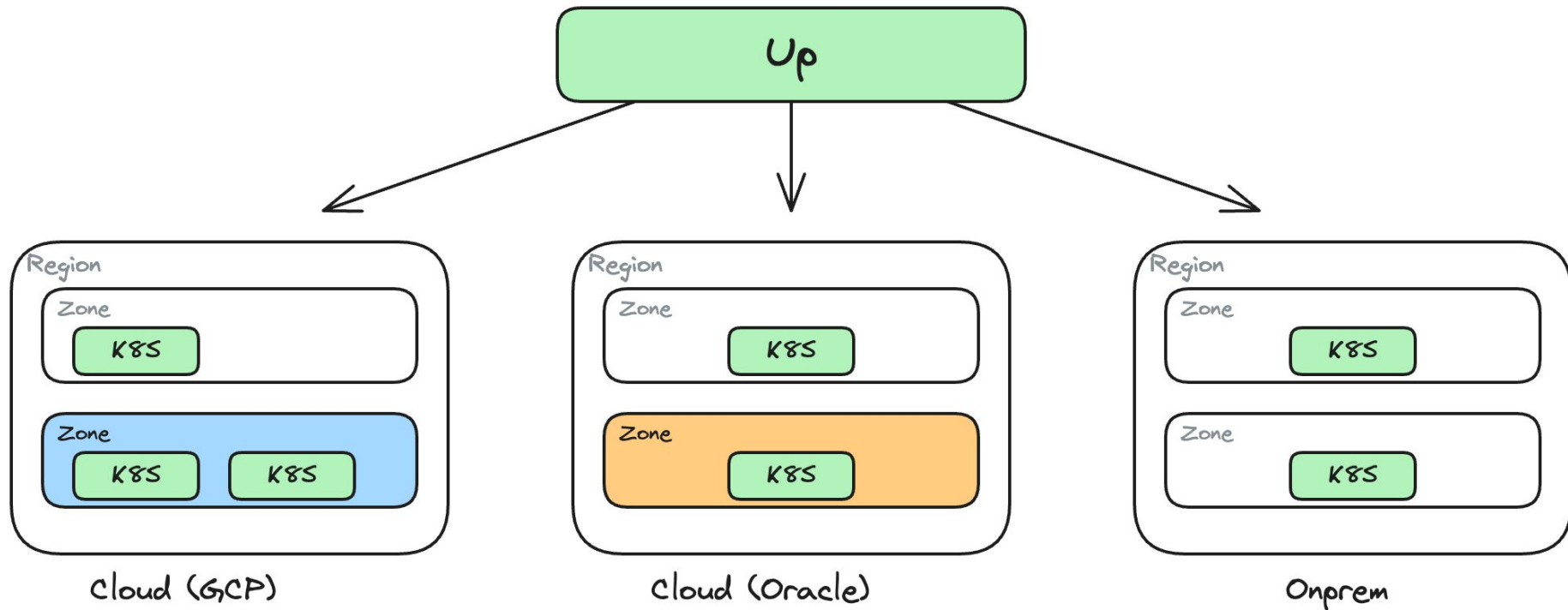
<uns://dca22/dca22-kubernetes-stateless01/us1/up-cd-test-service-012/staging/compute-0>[View DCA logs](#) [Find instances](#)[View instance state history graph for DCA](#)

2 INSTANCES RUNNING

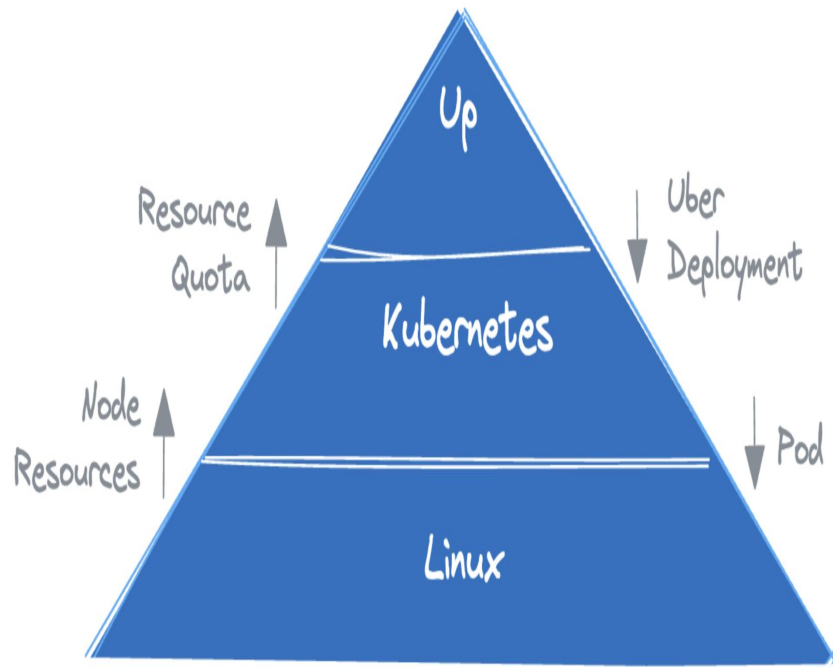
2 INSTANCES REQUESTED

Ongoing Operations

Stateless Compute Platform



Stateless Compute Platform

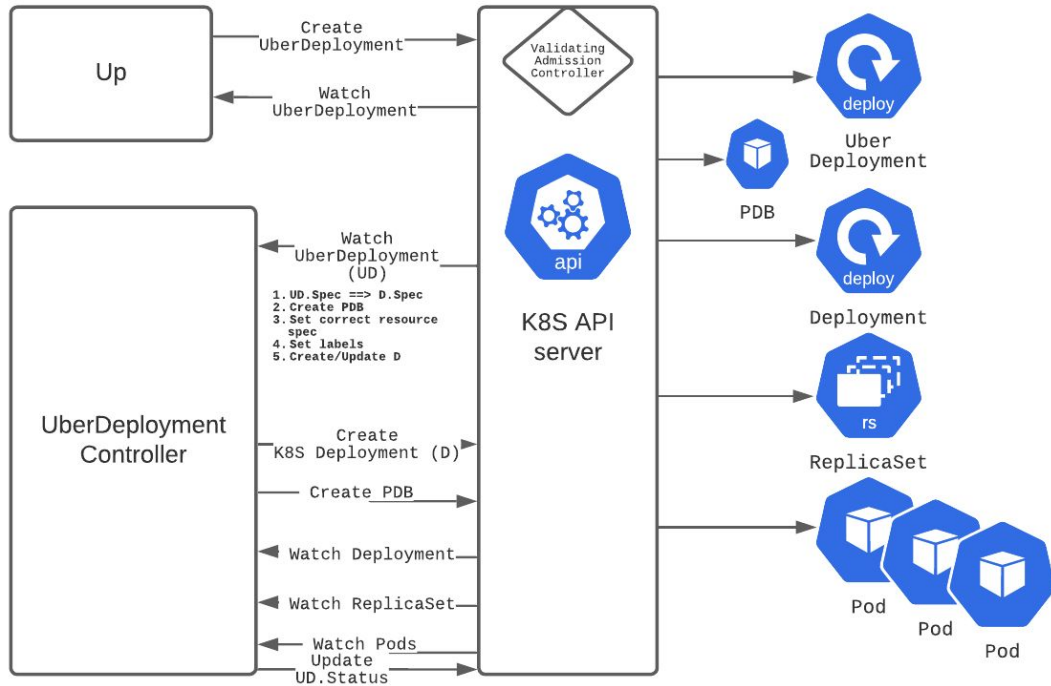


Uber Deployment Controller - UDC

Up communicates service intent to K8s via Uber Deployment, an inhouse CRD.

Uber Deployment Controller enforces this intent by reconciling changes to Uber Deployment - ultimately creates pods of the service while honoring a plethora of platform SLAs.

Encapsulates the K8s state into a single terminal workflow status field. Up's internal state machine relies on this field.



- Deployment
- Instance health
- Environment history
- Capacity
- Continuous Profiler
- Incident mitigation
- Networking
- Settings

Other resources

[Logs](#) [Healthline](#) [Statistics](#) [Service Metrics](#)

Deployment

Deploy

Hotfix

Restart

Current revisions [Show details](#)

027e57c



Prod. version of CD micro repo test service

Target revision

DCA

— Instances dca22: 2

dca22 > dca22-kubernetes-stateless01



2 running

<uns://dca22/dca22-kubernetes-stateless01/us1/up-cd-test-service-012/staging/compute-0>[View DCA logs](#) [Find instances](#)[View instance state history graph for DCA](#)

2 INSTANCES RUNNING

2 INSTANCES REQUESTED

Ongoing Operations

Regional failover as the last resort

2 active-active datacenters

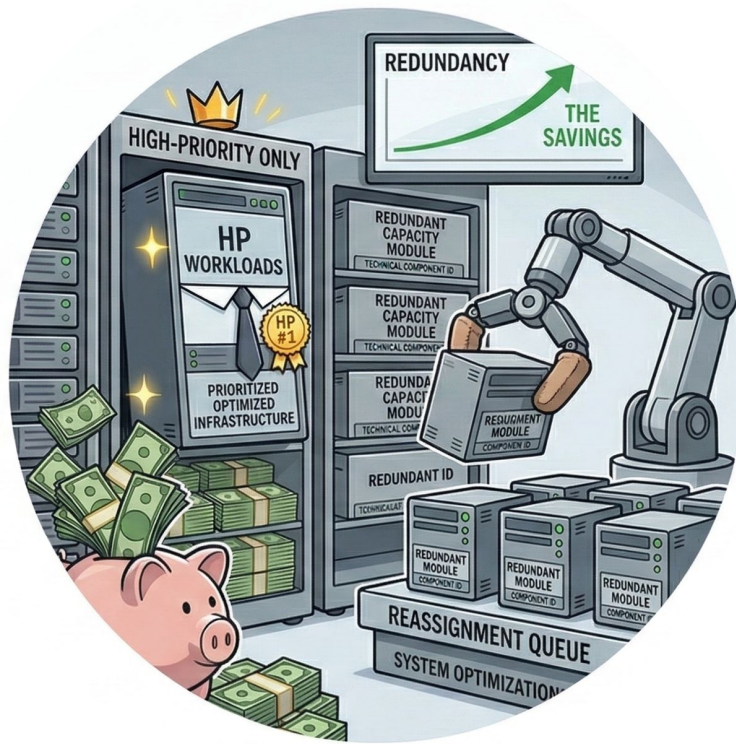
- Each handles some % of the user traffic
- Failover is about moving traffic from one to another
- Historically has 2x redundant capacity



The savings !!!!

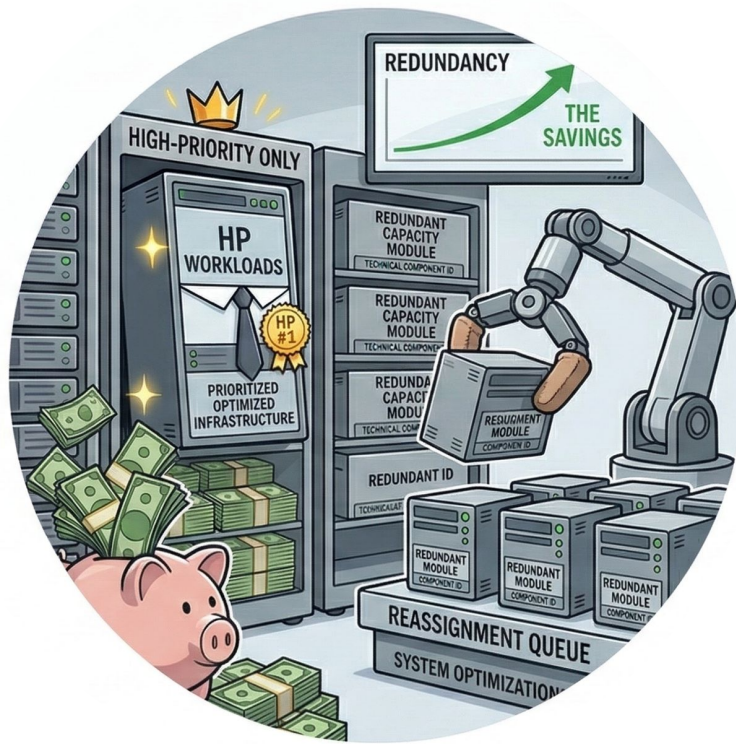
By reducing capacity redundancy

In the nutshell, the idea is simple: Let's terminate low-priority workloads and put the high-priority on their places.



How do we save money?

Replacing low-tiers with high-tiers



Uber is about scale

50+

Kubernetes clusters

1.5m

Pod launches per day

~10m

containers

Multiple Orchestrators

Usual path

aka UberDeployment. Uber operates as usual. That's the normal flow of the service lifecycle via Up.

Failover

Something is off with the datacenter. Low-tier workloads should be terminated and high-tier workloads should be placed instead.

HPA/Hybrid auto-scalers

Potential future with the multilayer horizontal scheduler, where a “clever” part sets borders and HPA behaves in them.

Is UDC not worthy?

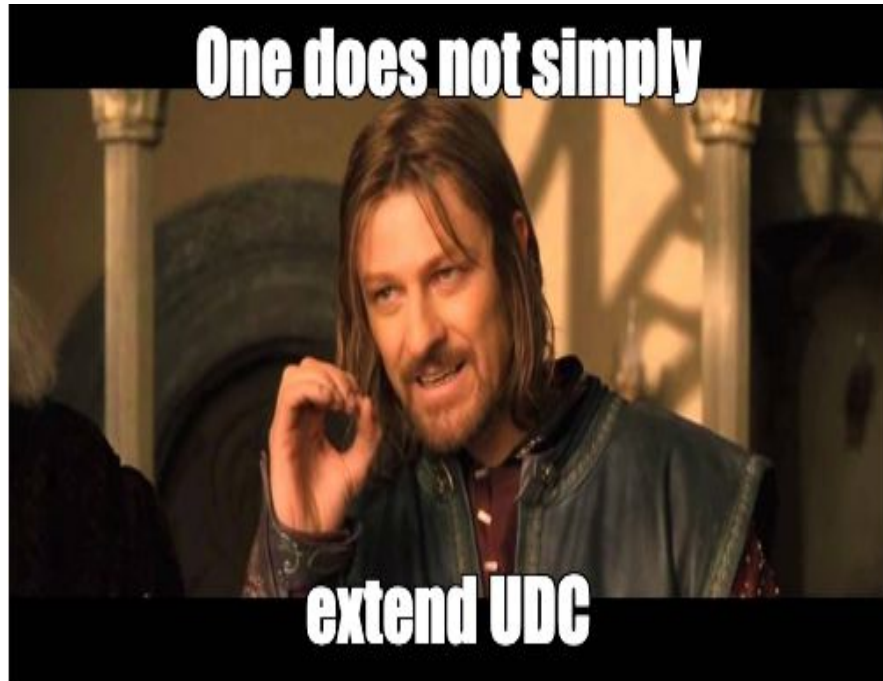
Too much power for a single controller

UDC was built with a single orchestrator in mind i.e Up. Fundamentally it only serves to enforce Up's requirements. It is not aware of other orchestrators and does not deign to dictate who gets to win.

UDC CRD design also reflects this

Hard to break an existing contract with millions of already running containers.

Up is a separate entity from Compute, in fact it is our customer. Extending UDC without changing Up is impossible.



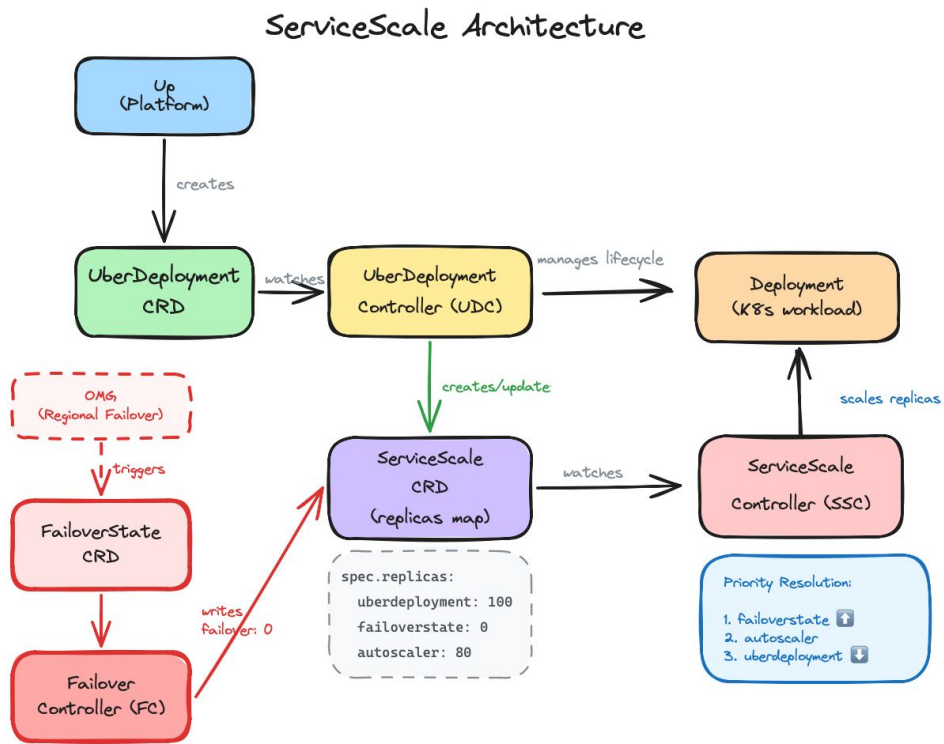


We need a new controller

ServiceScale CRD

```
kind: ServiceScale
meta:
  annotations:
    com.uber.compute/uber-deployment: test-ud # name of the Uber Deployment
    service: test-service                     # name of the Up service
spec:
  replicas:                                # map[string]*int
    uberdeployment: 10                     # Uberdeployment controller sets this value.
    failover: 0                            # The failover indicator
    autoscaler: 8                          # future
  policies:                               # scaling policies
    batch:
      size: 10
  scaleTargetRef:                          # crossVersionObjectReference ,points to a
workload resource
    apiVersion: apps/v1
    kind: Deployment
    name: winterfell
status:  # All the stories in a few slides ahead ;)
```

ServiceScale Controller



Status field

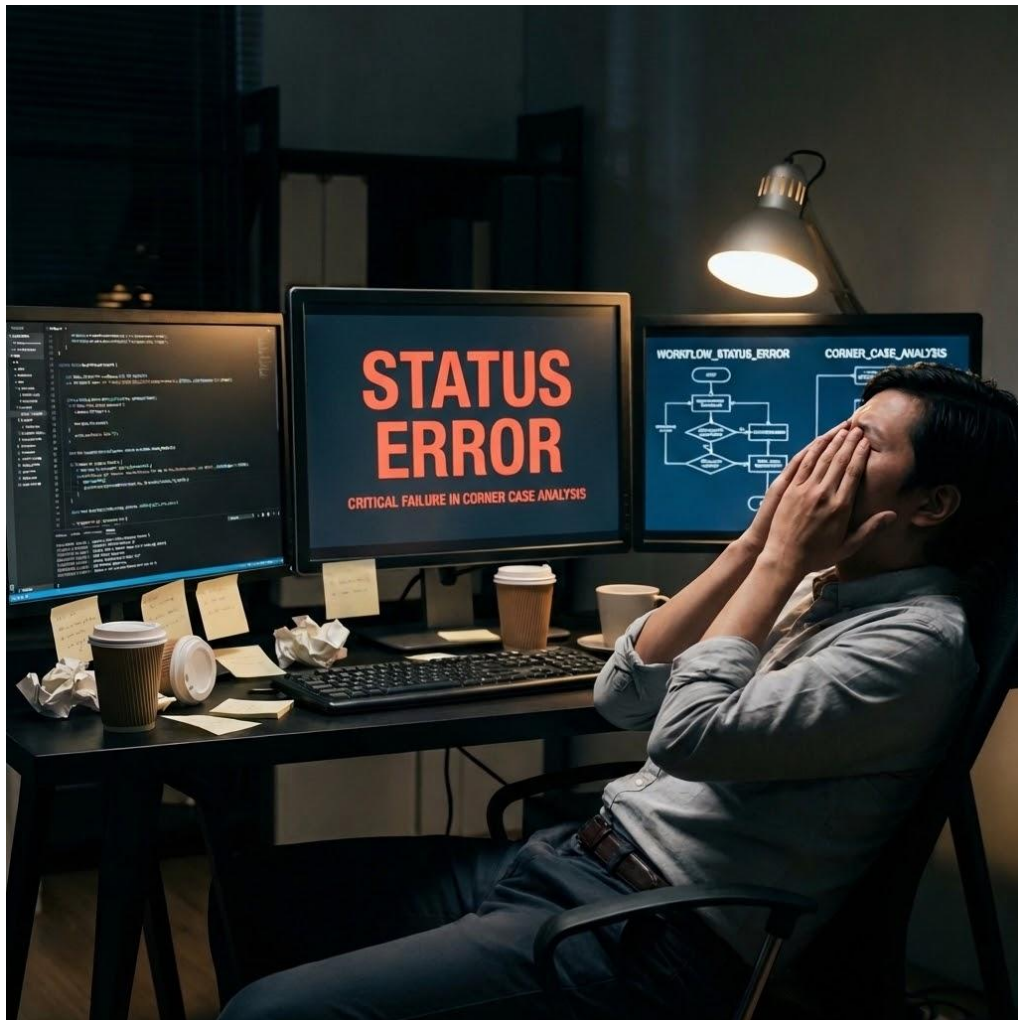
or the tough lesson
we learned

ServiceScale Controller

```
kind: ServiceScale
...
status:
  observedGeneration: 1
  controller: failover           # current scaling actor
  desiredReplicas: 0             # the replicas' count "goalstate"
  totalReplicas: 0               # count of replicas for any phase and any version
  readyReplicas: 0               # count of ready replicas on any version
  latestVersionReplicas: 0       # count of replicas on spec.version (any phase)
  workflowStatus: Success
  lastWorkflowTransition: 2024-04-03 08:00 # time when workflowStatus changed value last
```

ServiceScale Controller

```
kind: ServiceScale
...
status:
  observedGeneration: 1
  controller: failover           # current scaling actor
  desiredReplicas: 0             # the replicas' count "goalstate"
  totalReplicas: 0               # count of replicas for any phase and any version
  readyReplicas: 0               # count of ready replicas on any version
  latestVersionReplicas: 0       # count of replicas on spec.version (any phase)
  workflowStatus: Success
  lastWorkflowTransition: 2024-04-03 08:00 # time when workflowStatus changed value last
```



Terminal Status field

K8S has no “final” state. Any “final” state status field will be flaky.

K8s controllers read objects from caches

Computing a ‘terminal’ status field from stale objects is not a good idea.

We implemented our own ‘read your write’ guardrail before computing status to solve this



The two writer problem

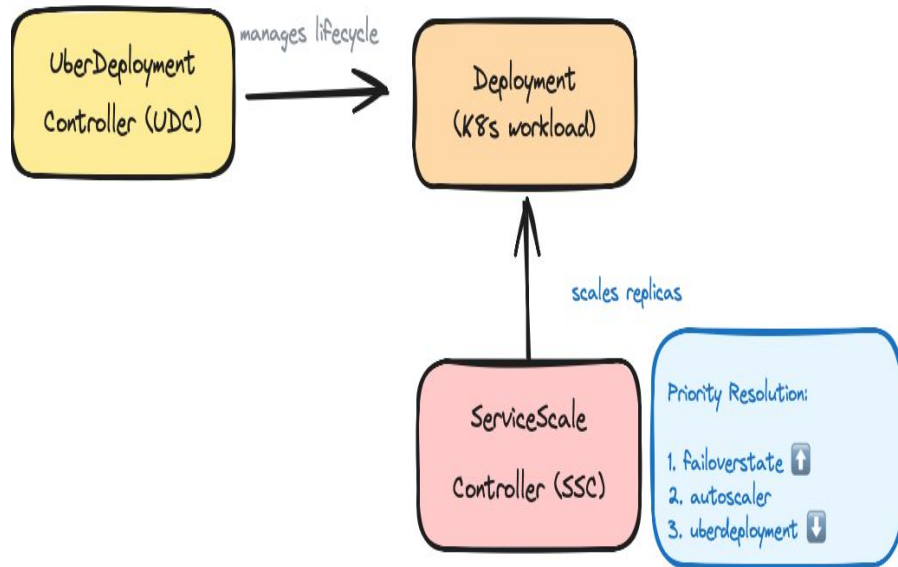
UDC and SSC update the underlying workload concurrently - the former via OCC Update and the latter via SSA patch to prevent races.

At scale, there were instances when controller manager drives the workload into a terminal inconsistent state requiring manual intervention to unblock.

The inconsistency manifests as annotation-spec drift for the latest replicaset managed by the deployment.

Controller manager source code has blind spots when it comes to handling simultaneous updates to the deployment resource.

Built custom logic to detect and mitigate such occurrences.



Uber