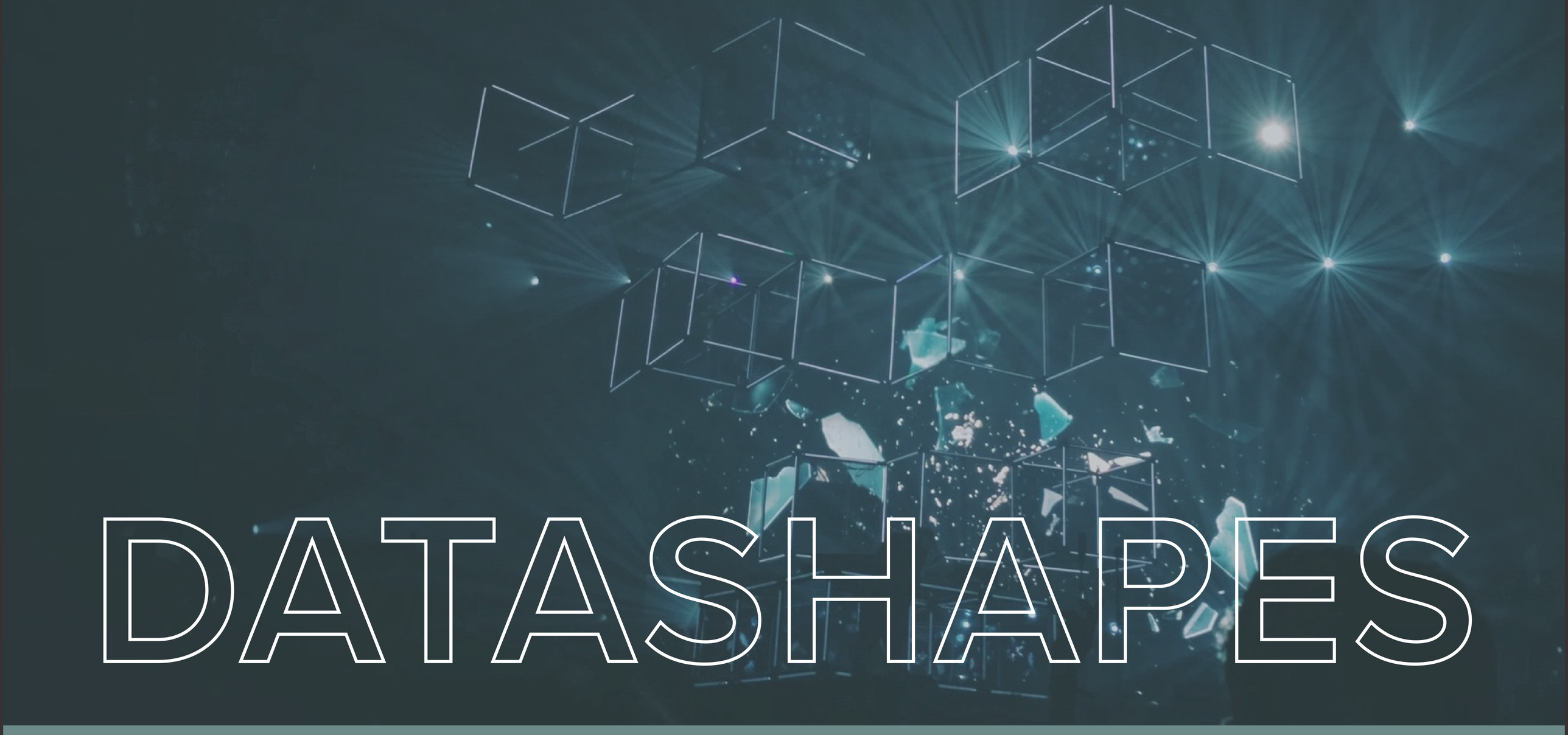
DataShapes



DATASHAPES

Highly-available multi-tenant production grade SaaS applications on Azure platform

CELESTIAL SYSTEMS

Project Overview

DataShapes case-study is about deploying multi-tenant SaaS applications to the production-grade Azure cloud Infrastructure with multi-tenant support. This same concept can be extended to other SaaS applications that may have similar use-case/requirements. The idea was to plan, design and implement an architecture that is highly-available, scalable and secure.

Know the Client:

DataShapes

A San Francisco Bay Area company, that is commercializing the patented, full-stack, market-ready DataShapes AI platform, in an enterprise-grade SaaS platform.

Business Requirements:

- Architect a cloud infrastructure design that can host and support SaaS applications and its tenants
- To build a scalable, high-available, and secure platform
- Support multi-tenant architecture
- The system should support autoscaling

Challenges:

The main challenge was the infrastructure resource provisioning. In order to avoid the wastage of resources, the resource allocation should be in such a way that, it should support multi-tenancy without having too many overheads in terms of resources.

Celestial Solution:

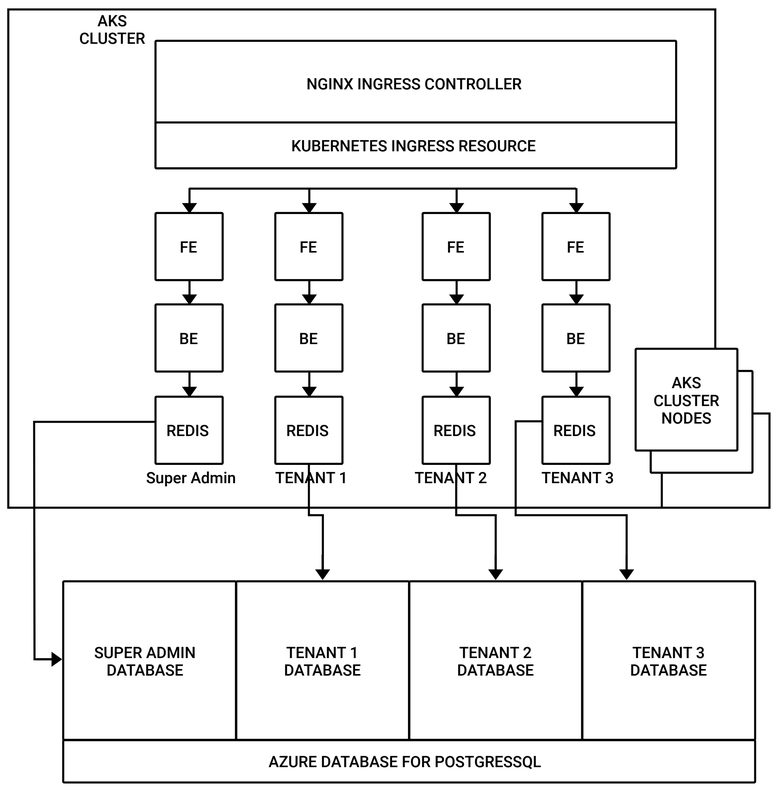
Celestial DevOps team assisted in planning, designing, and implementing the architecture that could support multi-tenant functionality of the SaaS application. We designed a solution based on Azure ecosystem in allocating dedicated resources for each new tenant onboarded.

Architecture:

Key Features:

Multi-tenancy:

The AKS service can have multiple services, deployments, and pods that could be scheduled across multiple VMs that are attached to this cluster. We have used the same concept to create separate and dedicated Kubernetes services, deployments and pods for each of the tenants. It means that for every tenant we can have distinct FE, BE and Redis instances running with the help of Kubernetes Services, deployments and pods objects.



Scalability:

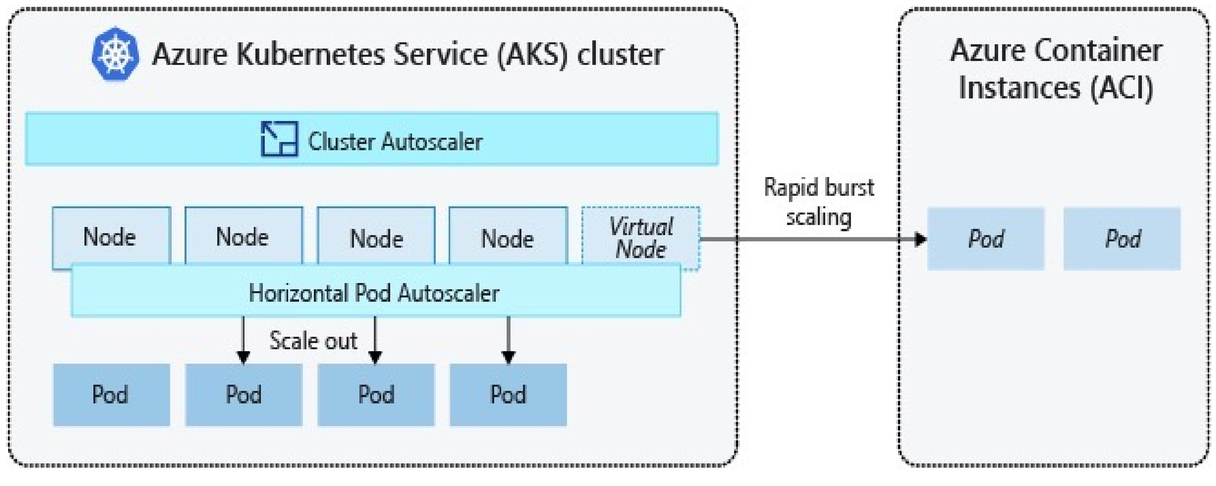
The architecture is equipped with auto-scaling at two levels: Cluster auto-scaler and Horizontal pod auto-scaler.

Horizontal Pod Auto-Scalar

If the load or any other metric as stated in the auto-scaling condition of pods is breached, then additional replicas of the pods will be spun to bridge the gap.

Cluster Auto-Scalar

If the overall load or any other metric as stated in the auto-scaling condition of node group (Virtual Machine Scale Set) is breached, then the additional node(s) will be launched to serve the traffic without any issues.



Img Source: https://docs.microsoft.com/en-us/azure/aks/concepts-scale

High-Availability:

The Kubernetes in-built feature will check the health of the pods and, if any of the pod crashes or gets terminated, the new corresponding pod will be scheduled. Likewise, if any node goes down, the AKS will spin up the new node and add it to the cluster, thus ensuring HA at all levels.

Security:

- There is only one entry point to the whole cluster via SaaS application and that is through the Azure Public Load Balancer. Rest all the other components, services, and nodes are private. They do not have public access.
- The connectivity is also established privately via spinning up helper pod from within the cluster, thus ensuring maximum security possible.
- The access to Kubernetes cluster API is controlled by Azure AD auth mechanism along with Kubernetes native RBAC mechanism. Both work in conjunction to ensure secure access control. User would have to get past both these authentications to obtain cluster access.

Onboarding of a new tenant from within the SaaS Application:

As per the architecture that we designed, the onboarding of a new tenant requires a dedicated set of Infra resources to be provisioned. In order to accomplish the same, the DevOps team had created a bash script. So, whenever the request to add a new tenant/org from the Web UI is placed, the corresponding Backend API is called, and it further executes the “deploy.sh” script. This runs the pre-defined checks and then, provisions the Infra resources (K8s objects – Services, deployments, pods) pertaining to that particular tenant/org for which the request is in question. After execution, it sends back the result status to the Backend API which further process it. This is how the automated provisioning of new tenants has been handled.

Error Handling in the Script:
The script would try to provision the Infrastructure resources required to successfully onboard a new tenant. If any of the events in the process failed, the script will stop the execution and delete the Infra resources that it had created by then.

Technologies Used:

DevOps Technologies:

Outcome:

- Implemented a Multi-tenant architecture
- Automated provisioning of Kubernetes objects
- Azure AD integration