Challenge 3 - I'm in the cloud, now what?

Background

Creating a cluster is just the first step for running containers in production, and it's important to think about operations and scenarios around your deployments. Areas such as telemetry and monitoring of clusters and services, state persistence, upgrades, and scale should all be planned for, and each come with their own unique challenges.

When taking your own existing/legacy applications and deploying them into the cloud, you want to have confidence that they're reliable, available, and tolerant to failures.

Challenge

Your team's goal in this challenge is to get your hands dirty with some real-world concerns by modifying the cluster and application you deployed in challenge 2. Your team will make that deployment 'production ready' by adding telemetry and monitoring, performing a successful container upgrade, adding reliable state to your container deployments, and scaling your cluster to multiple instances.

A Note on Containers and Minecraft State

State in containers by default is local to the container. As such, when the container is reset or recreated, the state goes with it forever. This is fine if your service or application is stateless, or only needs state during runtime, but not so much with persistent online worlds such as Minecraft. The typical solution? Connect the container to storage that exists outside of it, so that the state lives on long after any individual container is gone.

Minecraft server state is handled by a write-only operation to disk. The state is not verified by the server. This means that if two Minecraft servers share the same 'data' directory and use the default filename, they'll overwrite each other's data. No one will notice until the servers restart, leaving one server's data intact, and the other's lost forever (whichever server had an active player on it last, will win.)

One more thought: in the case where external storage is used and the servers are gone forever, how does that state get cleaned up?

Success Criteria

- Implement a monitoring/telemetry solution for your cluster and demonstrate it to a coach. This should include:
  - Memory usage per instance and or node.
  - CPU usage by instance or node.
  - Version of the container running on any given instance.
  - Number of container instances running.
  - Population of each Minecraft server.
  - Total current players and available capacity.
- Upgrade your instance within the cluster above, to v2.0
- Configure your container deployments so that the state is persisted between instance resets. In order for us to verify this, you must ensure that both the default port for Minecraft (25565) and the default port for RCON (25575) must be exposed and available publicly.

References

- You can find the Minecraft containers on Docker Hub
- HINT: Just a reminder the RCON port for minecraft is 25575, in **addition** to the default connection port (25565)
- HINT: Players **expect** that if they connect to the same server, the world is just as they left it and everything they built is still there (unless another player destroyed it).

Some other useful resources in addition to the ones in challenge 2 are:

- Azure resource naming best practices

- Azure CLI reference
- Kubernetes volumes
- Service Fabric Containers Overview