1. Function app

<https://medium.com/@robertdennyson/how-azure-functions-work-behind-the-scenes-provisioning-scaling-and-the-role-of-docker-and-d62e64255642>

**Scaling Mechanism: How Azure Functions Scales on Demand**

One of the most powerful aspects of Azure Functions is its ability to automatically scale in response to workload demands. Whether you’re processing a single HTTP request or millions of messages from a queue, Azure Functions can scale accordingly.

**1. Consumption Plan: Auto-Scaling Based on Events**

In the default **Consumption Plan**, Azure Functions uses an event-driven scaling model, where it provisions and scales function instances based on incoming triggers.

* **Automatic Scaling**: When a function is invoked, Azure monitors the demand (e.g., incoming HTTP requests or messages) and provisions new containers to handle the load. Each container hosts one or more function instances.
* **Scaling to Zero**: In the absence of traffic, Azure Functions can scale down to zero containers to minimize resource usage and cost. When new requests arrive, Azure dynamically provisions containers again.
* **Per-Function Scaling**: Each function is scaled independently. For example, if you have multiple functions in a function app, Azure will scale each based on its specific demand.

**2. Premium and Dedicated (App Service) Plans: More Control**

For scenarios where cold starts are a concern, or you require more control over scaling, Azure Functions also offers **Premium** and **Dedicated Plans**:

* **Always-On Instances**: In these plans, you can configure “always-on” instances to avoid cold starts. This ensures that at least one container is always available to serve requests immediately.
* **Manual Scaling**: These plans provide more control over scaling, allowing you to pre-provision resources and define how many instances you want running at any given time.

**Docker and Kubernetes Behind Azure Functions**

Now that we understand provisioning and scaling, let’s explore the technologies that power Azure Functions behind the scenes: **Docker** and **Kubernetes**.

**1. Docker: Containerization of Azure Functions**

Under the hood, Azure Functions leverages **Docker** to provide isolated, portable environments for function execution. Each function is packaged as a Docker container, ensuring the runtime environment is consistent across deployments.

* **Docker Containers**: When you deploy a function, Azure packages it along with the necessary runtime (e.g., .NET, Node.js, Python) and dependencies into a Docker container. This container provides isolation, making sure your function runs independently from other workloads.
* **Managed by Azure**: In most cases, Azure handles the entire lifecycle of these containers, including provisioning, scaling, and tearing them down when they’re no longer needed.

**2. Kubernetes: Orchestration in Advanced Scenarios**

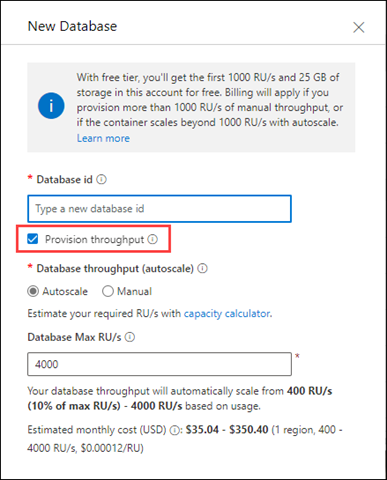
While Docker containers handle the execution of functions, **Kubernetes** (or Kubernetes-like orchestration) is used in more advanced scenarios for orchestrating these containers. Although not directly exposed in the default consumption plan, Kubernetes is involved in certain deployment environments:

* **Azure Functions on Kubernetes (KEDA)**: For more advanced deployments, such as running Azure Functions on **Azure Kubernetes Service (AKS)** or **Azure Arc-enabled Kubernetes**, **KEDA** (Kubernetes-based Event-Driven Autoscaler) is used. KEDA allows Kubernetes to scale function containers based on external event triggers (e.g., messages in a queue or HTTP traffic).
* **KEDA**: KEDA allows for automatic scaling of function pods (containers in Kubernetes) based on event triggers, such as HTTP requests, queue messages, or database changes. It ensures the Kubernetes cluster dynamically adjusts the number of running function instances in response to load.
* **Azure Arc**: Azure Functions can be extended to **on-premises or multi-cloud environments** using **Azure Arc-enabled Kubernetes**. In these environments, Kubernetes handles orchestration, while Azure Functions provides the serverless execution model on top of the cluster.

1. Cosmos DB

Azure Cosmos DB works with scaling throughput in two ways: provisioned throughput, which is used in the demo, and serverless.

At the time that you create your database, you must decide whether to use provisioned throughput or serverless scaling.



1. Blob Storage

Azure Blob Storage automatically scales to accommodate increasing data storage needs, with a limit of 5 PiB per storage account. To optimize performance and cost, consider factors like blob size, access tiers, and using a GPv2 storage account. For specific capacity or ingress limit increases beyond the default, contact Azure Support.

Scaling Up Blob Storage:

* + **Automatic Scaling:**

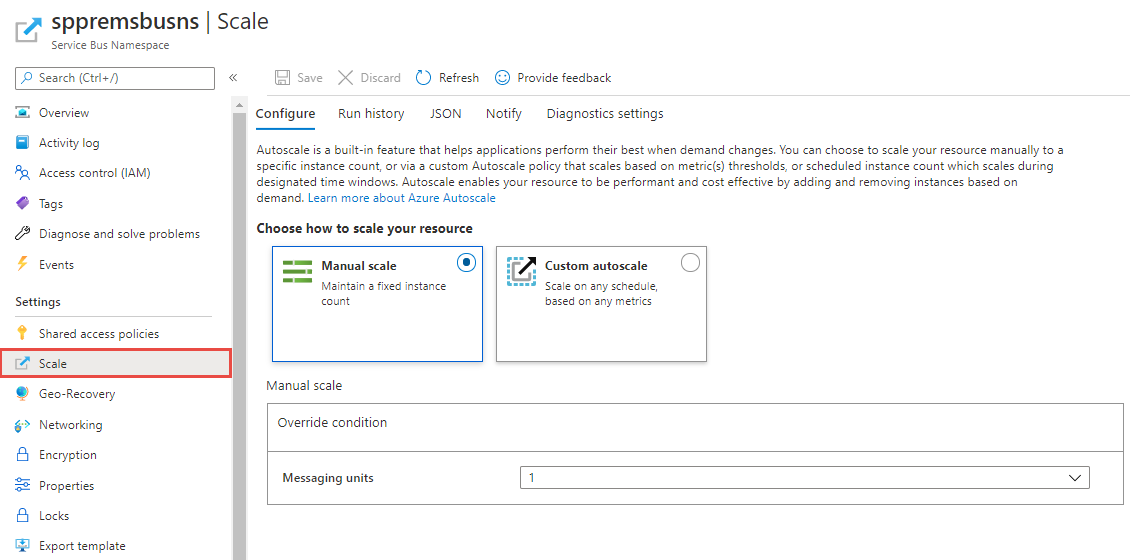
Azure Blob Storage is designed to scale automatically to meet your storage needs, with a limit of 5 PiB per storage account.

* + **No Provisioning:**

You don't need to pre-provision the size of your container; it scales as needed.

Service Bus Scaling

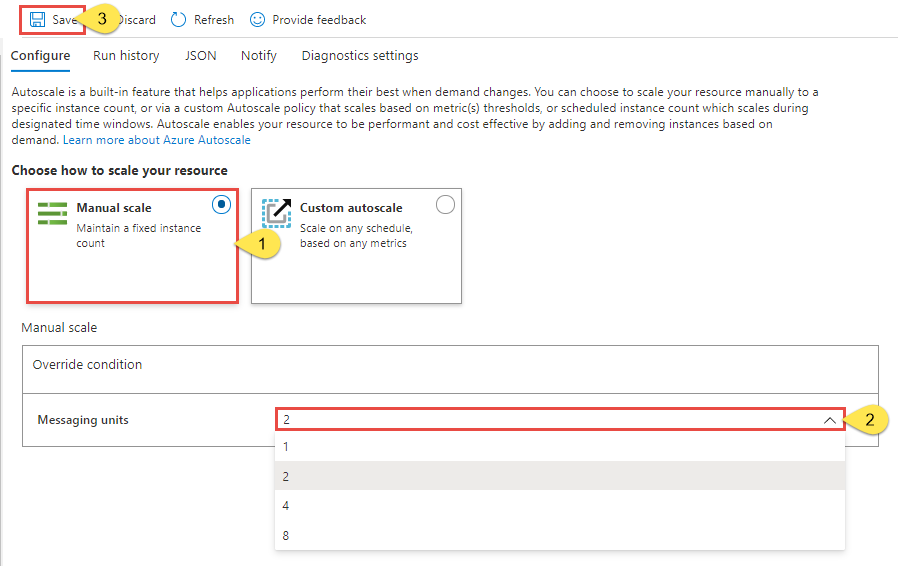
1. Sign in to the [Azure portal](https://portal.azure.com/).
2. In the search bar, type **Service Bus**, select **Service Bus** from the drop-down list, and press **ENTER**.
3. Select your **premium namespace** from the list of namespaces.
4. Switch to the **Scale** page.



**Manual scale**

This setting allows you to set a fixed number of messaging units for the namespace.

1. On the **Autoscale setting** page, select **Manual scale** if it isn't already selected.
2. For **Messaging units** setting, select the number of messaging units from the drop-down list.
3. Select **Save** on the toolbar to save the setting.



**Custom autoscale - Default condition**

You can configure automatic scaling of messaging units by using conditions. This scale condition is executed when none of the other scale conditions match. You can set the default condition in one of the following ways:

* Scale based on a metric (such as CPU or memory usage)
* Scale to specific number of messaging units