# AlloyDB Omni Installation Guide

## Table of contents


Table of contents 1
Introduction 2
- Architecture 3
  - AlloyDB Omni components 3
    - Columnar engine 3
    - Automatic memory management 4
    - Adaptive autovacuum 4
    - AlloyDB AI/ML worker 5
  - AlloyDB Omni versus AlloyDB for PostgreSQL service on Google Cloud 5
  - Advantages of running AlloyDB Omni as a container 5

Plan your AlloyDB Omni installation 6
- Size and capacity planning 6
- Prerequisites for running AlloyDB Omni 7
  - Hardware requirements 7
  - Software requirements 7
- Supported storage types 8
  - Local NVMe or SAN storage 8
  - Local NVMe storage 8
  - SAN storage 9
- Count and limit vCPUs 9
- Container engine selection and configuration 9
- SELinux 10

Installation 10
- Install AlloyDB Omni 10
  - Create file system location for AlloyDB Omni 10
  - Create an AlloyDB Omni container 12
- Connect directly to an AlloyDB Omni instance 13
  - Suggested parameter changes 14
- Change configuration parameters 15
- Install AlloyDB Omni with AlloyDB AI 16

# Introduction

AlloyDB Omni is a downloadable database software package that lets you deploy a streamlined edition of AlloyDB for PostgreSQL in your own computing environment. The portability of AlloyDB Omni lets it run in a wide variety of environments, including data centers, cloud-based VM instances, and laptops.

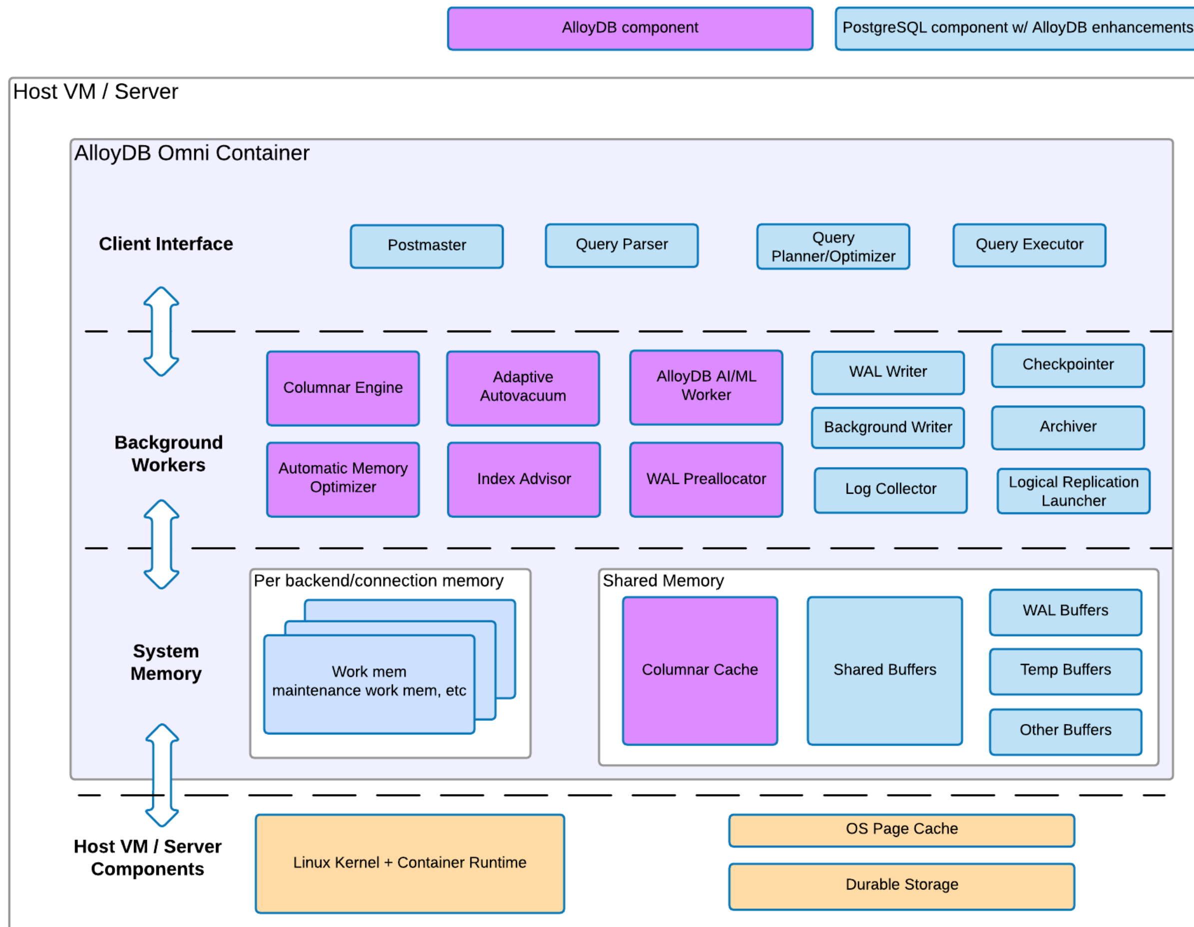
AlloyDB Omni is well-suited to the following scenarios:

- You need a scalable and performant version of PostgreSQL, but you can't run a database in the cloud due to regulatory or data-sovereignty requirements.
- You need a database that keeps running even when it's disconnected from the internet.
- You want to physically locate your database as close as possible to your users, in order to minimize latency.
- You want to migrate away from a legacy database without committing to a full cloud migration.
- You have workloads that need to run in multiple public clouds, or in hybrid environments with private data centers and public clouds intermixed.

AlloyDB Omni doesn't include AlloyDB features that rely on operation within Google Cloud. If you want to upgrade your project to the fully managed scaling, security, and availability features of AlloyDB, you can migrate your AlloyDB Omni data into an AlloyDB cluster like with any other initial data import.

AlloyDB Omni includes many of the performance improvements in AlloyDB for PostgreSQL. AlloyDB Omni has more than twice the throughput of open source PostgreSQL on TPC-C-like tests with HammerDB, and an improvement in response time by up to a hundred times for TPC-H-like analytical queries. AlloyDB Omni also supports autopilot features from AlloyDB in Google Cloud, allowing it to self-update and self-tune. These autopilot features include automatic memory management, adaptive autovacuum, and index advisor. AlloyDB also supports the columnar engine, which can accelerate analytical queries on large data sets by up to 100 times.

# Architecture




## AlloyDB Omni components

### Columnar engine

The AlloyDB columnar engine accelerates SQL query processing of scans, joins, and aggregates by providing these components:

- An in-memory column store that contains table and materialized-view data for selected columns, reorganized into a column-oriented format. By default this column store consumes 1 GB of available memory. You can change the amount of memory usable by the column store by setting the `google_columnar_engine.memory_size_in_mb` parameter in the postgresql.conf used by your AlloyDB Omni instance, see Change configuration parameters.
- A columnar query planner and execution engine to support the use of the column store in queries.

### Automatic memory management

The automatic memory manager continuously monitors and optimizes memory consumption across an entire AlloyDB Omni instance. This module adjusts the shared buffer cache size as necessary based on memory pressure as you run your workloads. By default, the automatic memory manager sets the upper limit to 80% of system memory and allocates 10% of system memory for shared buffer cache. You can configure an upper limit for the size of the shared buffer cache by setting the `shared_buffers` parameter in the postgresql.conf used by your AlloyDB Omni instance, see Automatic memory management for more information.

### Adaptive autovacuum

AlloyDB's adaptive autovacuum automatically adjusts the frequency of vacuuming and analyzes operations based on the workload of the database. This automatic adjustment helps the database run at peak performance, even as the workload changes, without any interruption from the vacuum process.

AlloyDB's adaptive autovacuum uses a number of factors to determine the frequency of vacuuming and analyze operations, which include the following:

- The size of the database
- The number of dead tuples in the database
- The age of the data in the database
- The number of transactions per second versus the estimated vacuum speed

The following are the adaptive autovacuum improvements and automatically adjusted settings in AlloyDB:

- **Dynamic vacuum resource management**: Instead of using a fixed cost limit, AlloyDB uses real-time resource statistics to adjust the vacuum workers. When the system is busy, the vacuum process and resources are throttled. If enough memory is available, additional memory is allocated for vacuum workers to accelerate index vacuum.
- **Dynamic XID Throttling**: AlloyDB automatically and continuously monitors the progress of vacuuming and the speed of transaction ID consumption. If a risk of transaction ID wraparound is detected, AlloyDB will begin to throttle transaction ID consumption by slowing down transactions. AlloyDB also allocates more resources to vacuuming so that vacuuming can catch up and return to the safe zone. During this process, the overall transactions per second are reduced until the transaction IDs are in the safe zone (observable as sessions waiting on "AdaptiveVacuumNewXidDelay"). When the transaction ID age increases, the vacuum workers are dynamically increased.
- **Efficient vacuuming for larger tables**: The default logic used to decide whether to vacuum a table is based on table-specific statistics stored in `pg_stat_all_tables`

which has the dead tuple ratio. That logic works for small tables but may not work efficiently for larger, frequently updated tables. AlloyDB has an updated scan mechanism that helps trigger the autovacuum more often, scanning chunks of large tables and helping remove dead tuples more efficiently.
- **Log warning messages**: In AlloyDB, the vacuum blockers, such as long-running transactions, orphaned prepared transactions, or orphaned replication slots, are detected and warnings are registered in the PostgreSQL logs so that users can handle the cases in a timely manner.

### AlloyDB AI/ML worker

The AlloyDB AI/ML worker is the core of AlloyDB AI in AlloyDB Omni. The worker provides all of the capabilities necessary for calling VertexAI models directly from the database. The AI/ML worker runs as a process called `omni ml worker`.

## AlloyDB Omni versus AlloyDB for PostgreSQL service on Google Cloud

AlloyDB Omni and the fully-managed AlloyDB for PostgreSQL service on Google Cloud share the same core components. The major difference between the two is the durable storage layer. The storage layer of AlloyDB in the cloud is built on top of Colossus, Google's distributed storage system, and writes are committed by asynchronous processes that avoid the overhead of the open source PostgresSQL checkpoint process. With AlloyDB Omni, you choose your storage system as you would with an open source PostgreSQL database and therefore require a technical team with system administration and DBA expertise. The architecture of AlloyDB in the cloud provides optimizations for increased WAL performance and read pools.

## Advantages of running AlloyDB Omni as a container

Google distributes AlloyDB Omni as a container that you can run with container runtimes such as Docker and Podman. Operationally, containers present the following advantages:
- **Transparent dependency management:** All necessary dependencies are bundled in the container and tested by Google to ensure they are fully compatible with AlloyDB Omni.
- **Portability:** You can confidently expect AlloyDB Omni to behave consistently across environments.
- **Better security isolation:** You choose what the AlloyDB Omni container has access to on the host machine.
- **Better resource management:** You can easily define the amount of compute resources the AlloyDB Omni container should use.

- **Seamless patching and upgrades:** Patching a container consists of simply replacing the old image with a new one.

# Plan your AlloyDB Omni installation

## Size and capacity planning

Sizing a PostgreSQL environment is a multifaceted process that involves considering several factors to ensure optimal performance, reliability, and cost-effectiveness. When migrating an existing database the CPU and memory resources required for AlloyDB Omni are similar to the requirements of the source database system. Plan to start with a deployment using matching CPU, RAM and disk resources, as well as using the source system configuration as the AlloyDB Omni baseline configuration. You might be able to reduce your resource consumption after performing sufficient testing. Here's a breakdown of the key steps:

1. Define your workload:

- **Data volume**: Estimate the total amount of data you'll store in PostgreSQL. Consider both the current data and projected growth over time.
- **Transaction rate**: Determine the expected number of transactions per second (TPS), including reads, writes, updates, and deletes.
- **Concurrency**: Estimate the number of concurrent users or connections accessing the database.
- **Performance requirements**: Define your acceptable response times for different types of queries and operations.

2. Hardware considerations:

- **CPU**: AlloyDB Omni benefits from multiple CPU cores, linearly scaling to 64 cores, whereas open source PostgreSQL generally does not benefit from greater than 16 vCPUs. Consider the number of cores based on your workload's concurrency and computation needs. Take into consideration any gains that might be present due to a change in CPU generation or platform.
- **Memory**: Allocate sufficient RAM for PostgreSQL's shared buffers for caching data and working memory for query processing. The exact requirement depends on the workload. Start with 8 GB of RAM per vCPU.
- **Storage**:
  - **Type**: Choose between local NVMe storage for highest performance or SAN storage for scalability and data sharing, based on your requirements.

    - **Capacity**: Ensure enough storage for your data volume, indexes, WAL (Write-Ahead Log), backups, and future growth.
    - **IOPS**: Estimate the required input/output operations per second (IOPS) based on your workload's read and write patterns. When running AlloyDB Omni in a public cloud, consider the performance characteristics of your storage type to understand if you need to increase storage capacity to meet a specific IOPS target.

## Prerequisites for running AlloyDB Omni

### Hardware requirements

| OS/Platform | Minimum hardware requirements | Recommended hardware configuration |
|---|---|---|
| **Linux** | • x86-64 or ARM CPU with AVX2 support<br>• 2 GB of RAM<br>• 10 GB of disk space | • x86-64 or ARM CPU with AVX2 support or ARM CPU<br>• 8GB of RAM for every vCPU allocated to AlloyDB Omni<br>• 20+ GB of disk space |
| **MacOS** | • Intel CPU with AVX2 support or M-chip<br>• 2GB of RAM<br>• 10 GB of disk space | • Intel CPU with AVX2 support or M-chip<br>• 8GB of RAM for every vCPU allocated to AlloyDB Omni<br>• 20+ GB of disk space |

- To install AlloyDB Omni on a cloud platform, we recommend using the following instance types:
    - On Google Cloud, we recommend n2-highmem instances.
- We recommend that you use a dedicated solid-state drive (SSD) storage device for storing your data. If you use a physical device for this purpose, we recommend attaching it directly to the host machine.

### Software requirements

| OS/Platform | Minimum software requirements | Recommended software configuration |
|---|---|---|
| **Linux** | • Debian based OS (Ubuntu, etc) or RHEL 9 | • Debian based OS (Ubuntu, etc) or RHEL 9<br>• Linux kernel version 6.1 or higher or any |

| | | |
|---|---|---|
| | • Linux kernel version 5.3 or higher<br>• Cgroupsv2 Enabled<br>• Docker Engine 20.10+ or Podman 4.2.0+ | Linux kernel version older than 5.3 that has support for the MADV_COLLAPSE and MADV_POPULATE_WRITE directives.<br>• Cgroupsv2 Enabled<br>• Docker Engine 25.0.0+ or Podman 5.0.0+ |
| **MacOS** | • Docker Desktop 4.20 or higher | • Docker Desktop 4.30 or higher |

AlloyDB Omni supports the following operating systems:

- RedHat 9
- Ubuntu 22.04+
- Debian 11 and 12
- MacOS

## Supported storage types

AlloyDB Omni supports file systems on block storage volumes within database instances. Smaller development or trial systems may prefer to use the local file system of the host where the container is running, but enterprise workloads should use storage that is reserved for AlloyDB Omni instances. These storage devices can either be configured in a singleton configuration with one disk device for each AlloyDB Omni container, or a consolidated configuration where multiple AlloyDB Omni containers read and write from the same disk device. This choice depends on the demands set by your database workloads.

### Local NVMe or SAN storage

Both local Non-Volatile Memory Express (NVMe) storage and Storage Area Network (SAN) storage offer distinct advantages. Choosing the right solution depends on your specific workload requirements, budget, and future scalability needs.

The best choice depends on your needs:

- To prioritize absolute performance, choose local NVMe.
- If you need large-scale, shared storage, choose a SAN.
- If you need to balance performance and sharing, consider a SAN with NVMe over Fabrics for faster access.

### Local NVMe storage

NVMe is a high-performance protocol designed for solid-state drives (SSDs). NVMe SSDs connect directly to the PCIe bus, bypassing traditional storage controllers, to deliver incredibly fast read and write speeds. Local NVMe storage provides the lowest latency and

highest throughput of any storage option. These reasons make local NVMe storage ideal for applications that demand extremely fast data access.

Scaling local NVMe storage typically involves adding more drives to individual servers. Adding more drives to individual servers can lead to fragmented storage pools and potential management complexities. Local NVMe storage is not designed for easy data sharing between multiple servers. Since local NVMe storage is local, server administrators must protect against disk failures using hardware or software Redundant Array of Inexpensive Disks (RAID). Otherwise, the failure of a single NVMe device will result in data loss.

### SAN storage

A SAN is a dedicated storage network that connects multiple servers to a shared pool of storage devices, often SSDs or centralized NVMe storage. While not as fast as local NVMe, modern SANs–especially those using NVMe over Fabrics–can still deliver excellent performance for most enterprise workloads.

- **SANs are highly scalable.** You can easily add more storage capacity or performance by adding new storage arrays or upgrading existing ones. SANs typically provide redundancy at the storage layer, providing protection against storage media failures.
- **SANs excel at data sharing.** Multiple servers can access and share data stored on the SAN, making it ideal for enterprise environments that require high availability. In the event of a server failure, SAN storage can be presented to another server in the datacenter, allowing for faster recovery.

## Licensing

The free AlloyDB Omni Developer edition can be used to evaluate, prototype, test, develop, and demonstrate your application, prior to production. Google licenses production deployments of AlloyDB Omni.

## Container engine selection and configuration

AlloyDB Omni is supported on both Docker and Podman, both of which can be run as root (rootful) or in rootless mode, where the daemon and containers run as a non-root user.

The advantages of rootful containers are simplicity, optimal network performance, and increased security by mitigating against the risk of potential vulnerabilities in the daemon and the container runtime. The most appropriate mode for your environment depends on your requirements and preferences.

The following instructions cover deploying a rootful AlloyDB Omni container.

## SELinux

This document assumes that SELinux, when present, is configured on the AlloyDB host to permit the AlloyDB Omni container to run, including access to the file system to be used by AlloyDB Omni (or SELinux is set to permissive).

# Installation

You can follow these steps to install and configure a single-instance setup of AlloyDB Omni. If you are interested in a multi-instance (high availability) configuration of AlloyDB Omni, see the *High Availability Installation* section of the AlloyDB Omni Configuration Guide.

## Install AlloyDB Omni

### Create file system location for AlloyDB Omni

Prior to creating an AlloyDB Omni instance you should decide where the AlloyDB Omni instance system and data will be stored.

When using an existing file system create a directory for use by the AlloyDB Omni container (as root):

```
Unset
mkdir -p /alloydb/<CONTAINER_NAME>/data
```

Replace the following:

`<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-1`.

When using a dedicated device perform the following steps:

1. Create a directory on the host for where the disk will be mounted (as root):

```
Unset
    mkdir -p /alloydb/<CONTAINER_NAME>
```

   Replace the following:

   `<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-1`.

2. Create a gpt partition table with a single partition on the disk device (as root):

```
Unset
    parted <DISK_DEVICE> mklabel gpt
    parted <DISK_DEVICE> mkpart primary 0% 100%
```

   Replace the following:

   `<DISK_DEVICE>`: The device name assigned by the operating system to the disk.

3. Create a file system on the disk device. We recommend using the `ext4` file system for AlloyDB Omni (as root).

```
Unset
    mkfs.ext4 -m 1 -L <CONTAINER_NAME> -F <DISK_PARTITION>
```

   Replace the following:

   `<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-1`. This entry is used to provide a label for the file system. The maximum length of an `ext4` file system label is 16 characters.

   `<DISK_PARTITION>`: The device name for the disk partition you will use to store the container's data.

4. Mount the device and create an entry in the /etc/fstab file so that the disk is mounted after a reboot (as root):

```
Unset
    echo -e
    "LABEL=<CONTAINER_NAME>\t/alloydb/<CONTAINER_NAME>\text4\tdefaults\t0 0"
    | tee -a /etc/fstab
    mount /alloydb/<CONTAINER_NAME>
    systemctl daemon-reload
```

   Replace the following:

   `<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-1`.

5. Create a data directory in the container specific file system (as root):

```
Unset
      mkdir /alloydb/<CONTAINER_NAME>/data
```

### Create an AlloyDB Omni container

(as root)

```
Unset
docker run --name <CONTAINER_NAME> \
    -e POSTGRES_PASSWORD=<NEW_PASSWORD> \
    -v /alloydb/<CONTAINER_NAME>/data:/var/lib/postgresql/data \
    -p <HOST_PORT>:5432 \
    -d google/alloydbomni
```

For RHEL-based distributions, select [docker.io/google/alloydbomni:latest]() from the list of images.

Replace the following:

`<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-1`.

`<NEW_PASSWORD>`: The password assigned to the new container's `postgres` user after its creation.

`<HOST_PORT>`: The TCP post on the host machine that the container should publish its own port 5432 to. To use the PostgreSQL default port on the host machine as well, specify `5432`.

## Connect directly to an AlloyDB Omni instance

You can connect directly to your AlloyDB Omni instance on the local host using the following command (as root):

```
Unset
docker exec -it <CONTAINER_NAME> psql -h localhost -U postgres
```

Replace the following:

<CONTAINER_NAME>: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, my-omni-1.

You can also connect to the database using the psql command on a client host. You are asked to enter the password of the postgres account when connecting. This password was provided when the container was created with the docker run command.

```
Unset
psql -U postgres -p <HOST_PORT> -h <IP_ADDRESS_OR_FQDN>
```

Replace the following:

<HOST_PORT>: The TCP post on the host machine that forwards to the container's AlloyDB Omni database.

<IP_ADDRESS_OR_FQDN>: The IP address or fully-qualified domain name of the host where AlloyDB Omni is running

## Suggested parameter changes

While parameters can vary greatly depending on the installation, in general, consider adjusting the following parameters from the default value. This is due to the fact that PostgreSQL is used for many purposes and the default parameters are not suitable for all applications.
The following parameters and their values are initial suggestions for modification from the default:

| Parameter | Suggested Value | Description |
|---|---|---|
| checkpoint_timeout | 1200 | Allows for longer times between checkpoints. |
| max_wal_size | 20480 | 20 GB of WAL logs can be written before a checkpoint occurs. Every checkpoint causes a full page write and imposing a reasonable checkpoint length reduces the number of full page writes from occurring. |
| min_wal_size | 10240 | Allows the pg_wal directory to shrink to this configuration. Allows reservation of space and also re-use of existing wal log files. |
| random_page_cost | 1.1 | Postgres default value of 4 is only appropriate for non-SSD type disks. |

| temp_buffers | 16384 | Sets memory for temp tables and common table expressions to 128MB. Further adjustment might be needed depending on VM size. |
|---|---|---|
| work_mem | 131072 | Sets memory for hash joins and sorts to 128MB. |
| max_worker_processes | 64 | |
| max_parallel_workers | 75% of vCPU count | Rounded to nearest whole even number. |
| max_parallel_workers_per_gather | 4 | Values higher than 4 generally produce less performance gains. |
| maintenance_work_mem | 2097152 | Set to 2 GB (2097152) if using a 16 GB instance or larger. Set to 1 GB (1048576) on instances of less than 16 GB memory. |
| max_parallel_maintenance_workers | 4 | |
| max_prepared_transactions | 500 | |
| default_toast_compression | lz4 | More performant than the default toast compression with minimal sacrifice of space. |
| alloydb.enable_auto_explain | on | Set this to **on** to avoid a restart. |
| auto_explain.log_min_duration | -1 | Min duration of -1 turns the auto explain off, but will allow a user to set in a session should auto explain need to be used. |
| auto_explain.log_buffers | on | |
| auto_explain.log_nested_statements | on | |
| auto_explain.log_settings | on | |
| auto_explain.log_triggers | on | |
| auto_explain.log_verbose | on | |
| auto_explain.log_wal | on | |
| alloydb.enable_pg_hint_plan | on | Set this to **on** to avoid a restart. |

## Change configuration parameters

To modify configuration parameters or add new configuration parameters update `postgresql.conf` and restart the container as follows:

1. Edit `/alloydb/<CONTAINER_NAME>/data/postgresql.conf`.
2. Restart `<CONTAINER_NAME>` (as root):

```
Unset
docker restart <CONTAINER_NAME>
```

Replace the following:

`<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-1`.

An example of this is covered in the following section.

## Install AlloyDB Omni with AlloyDB AI

Follow these steps if you intend to generate embeddings within the database by calling out to Vertex AI or other AI models. These instructions are not necessary if you plan to use `pgvector` or `pg_scann` vector indexes.

1. Create a service account with Google Cloud.
2. Create a service account key, in JSON format, and download it.

   Note: Service account keys are a security risk if not managed correctly.

3. Enable Vertex AI API in the project using the following:

```
Unset
    gcloud services enable aiplatform.googleapis.com
```

4. Add Vertex AI Identity and Access Management (IAM) permissions to the appropriate project and service account:

```
Unset
     gcloud projects add-iam-policy-binding <PROJECT_ID> \
         --member="serviceAccount:<SERVICE_ACCOUNT_ID>" \
         --role="roles/aiplatform.user"
```

Replace the following:
`<PROJECT_ID>`: the ID of your Google Cloud project.

`<SERVICE_ACCOUNT_ID>`: the ID of the service account that you created in the previous step, including the full `@PROJECT_ID.iam.gserviceaccount.com` suffix. For example: `my-service@my-project.iam-gserviceaccount.com`.

5. Copy the key to `/alloydb/<CONTAINER_NAME>/private-key.json`.
6. Adjust the file system permissions of the key (as root):

```
Unset
     chown $(docker exec <CONTAINER_NAME> id -u postgres):root
     /alloydb/<CONTAINER_NAME>/private-key.json
     chmod 600 /alloydb/<CONTAINER_NAME>/private-key.json
```

7. If an existing container of the same name exists, stop and remove it (as root):

```
Unset
     docker stop <CONTAINER_NAME>
     docker rm <CONTAINER_NAME>
```

Replace the following:

`<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-1`.

8. Start a new AlloyDB Omni container mounting the key into the container (as root):

```
Unset
docker run --name <CONTAINER_NAME> \
  -e POSTGRES_PASSWORD=<NEW_PASSWORD> \

  -p <HOST_PORT>:5432 \
 -v /alloydb/<CONTAINER_NAME>/data:/var/lib/postgresql/data \
  -v
"/alloydb/<CONTAINER_NAME>/private-key.json":/etc/postgresql/private-key.json \
```

```
-d google/alloydbomni
```

Replace the following:

`<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-ai-1`.

`<NEW_PASSWORD>`: The password assigned to the new container's `postgres` user after its creation.

`<HOST_PORT>`: The TCP post on the host machine that the container should publish its own port 5432 to. To use the PostgreSQL default port on the host machine as well, specify `5432`.

9. Update AlloyDB Omni by adding the following configuration options (as root):

Unset
```
echo "omni_enable_ml_agent_process = 'on'
omni_google_cloud_private_key_file_path = '/etc/postgresql/private-key.json'" \
| tee -a /alloydb/<CONTAINER_NAME>/data/postgresql.conf
```

Replace the following:

`<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-1`.

10. Restart AlloyDB Omni container (as root):

Unset
```
    docker restart <CONTAINER_NAME>
```

Replace the following:

`<CONTAINER_NAME>`: The name to assign this new AlloyDB Omni container in your host machine's container registry, for example, `my-omni-1`.