But First, Containers

Containers have been adopted as a great way to alleviate portability issues. Containers form the core of this OpenHack and underpin everything you'll be exploring as you progress through the challenges.

The objective of this challenge is to ensure you understand the very basics of containers, can work with them locally, and push them to a container image repository.

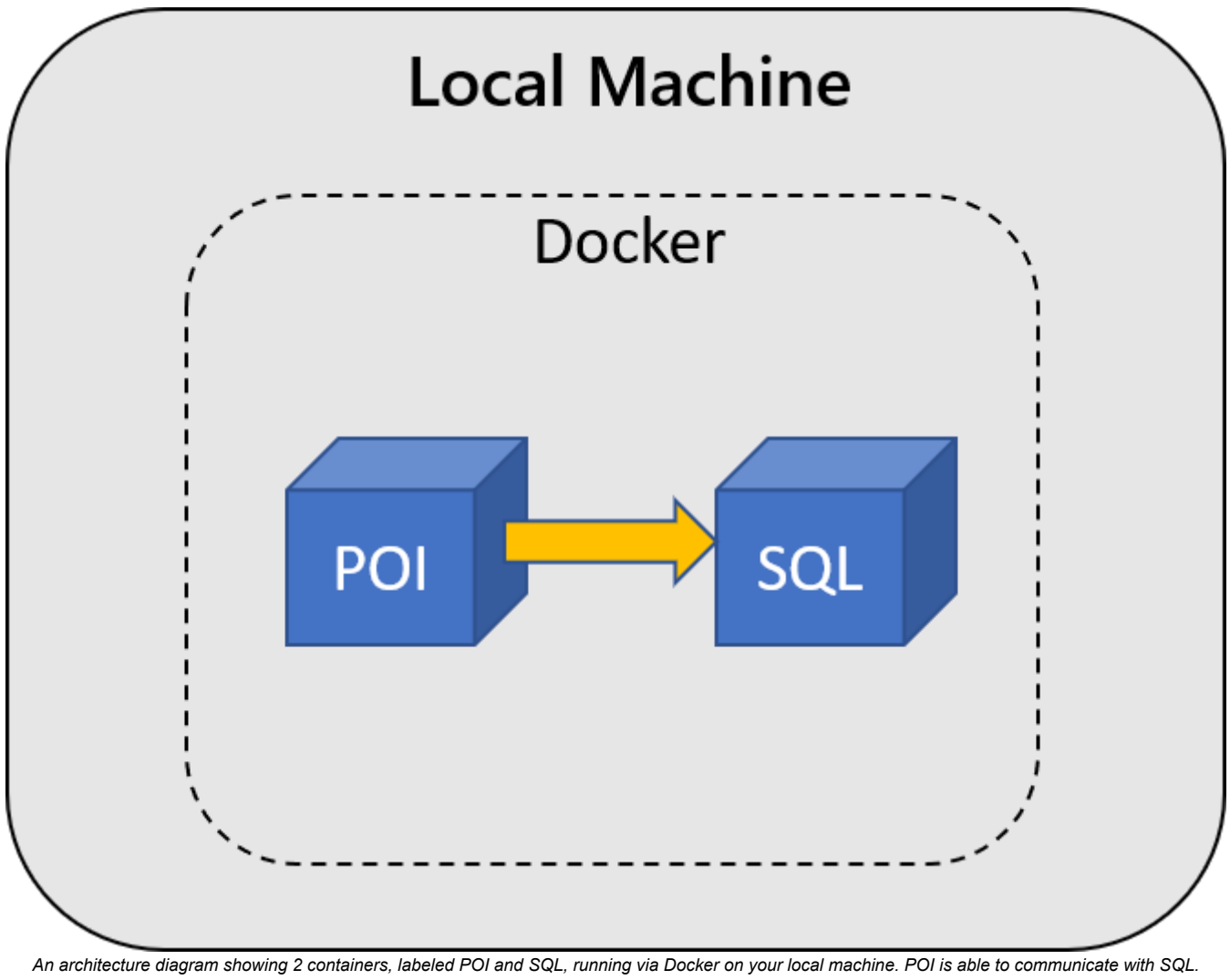
Challenge

You have been tasked with improving the local development experience for new developers by using Docker to simplify the building, testing, and running of the application. The CTO would also like to see this become part of the integration testing solution of the build pipeline.

Some of the work has been done for you, but it was during a time when teams were split between operations and development, leaving the code split between multiple codebases. The new CTO believes teams should be a mix of both Ops and Dev and has formed the team you are in now (say hi to your fellow teammates at your table :-)).

Building and Testing

Since you're new to this code base, you're going to verify at least one of the services still works by building and testing locally. In order to do this, you will need to build and run the Points of Interest (POI) container as well as a SQL Server container. The POI container communicates with the SQL Server container over the Docker network (https://docs.docker.com/v17.09/engine/userguide/networking):



An architecture diagram showing 2 containers, labeled POI and SQL, running via Docker on your local machine. POI is able to communicate with SQL.

> **Tip**: If you're having trouble matching the `Dockerfile` to the source code, remember the services are written in different languages. Take a look inside the `Dockerfile`, the corresponding service is more obvious than you think.

After setting up a SQL Server container running locally, use an image found in your team's Azure Container Registry (ACR) (already deployed to your Azure Subscription) in order to add sample data to the database. You'll need to authenticate to the registry first - reference the Azure Container Registry resource in the Azure portal for registry credentials.

```sh
docker run --network <networkname> -e SQLFQDN=<servername> -e SQLUSER=<db-user> -e SQLPASS=<password> -e SQLDB=mydrivingDB
<your-registry>.azurecr.io/data-load:1.0
```

> **Tip**: The `dataload` image used in the above command expects a database called "mydrivingDB" to exist already in SQL. Find a way to connect to your running SQL container to create this.

Then configure the POI application to connect to this SQL Server so you can test that the application works. You can find the curl commands (https://github.com/Microsoft-OpenHack/containers_artifacts/tree/main/src/poi#testing) to test the applications endpoints in the POI applications README.

> IMPORTANT: Set the `ASPNETCORE_ENVIRONMENT` environment variable in POI to `Local`. This configures the application to skip the use of SSL encryption, allowing connection to the local sql server.

Building and Pushing TripInsights Images

Now that you are sure the POI application works, the team must ensure that all of the TripInsights components are built as Docker images and pushed to the team's Azure Container Registry (ACR).

If you choose to test the rest of the images, you can run them locally and send an HTTP GET request to the health endpoint. For example, to hit the POI health endpoint on a container running locally on port 8080, curl or visit in-browser http://localhost:8080/api/poi/healthcheck (http://localhost:8080/api/poi/healthcheck). Endpoints other than the health endpoint may not be functional at this point (due to dependencies on APIs or the SQL database), so don't worry if you can't reach them.

Success Criteria

- **Each member** of your team must show your coach a locally running Points of Interest (POI) container connected to a running SQL Server container. Verify that your POI container is serving content via HTTP commands (https://github.com/Microsoft-OpenHack/containers_artifacts/tree/main/src/poi#testing). Explain your setup to your coach and how it could be used for development and testing.
- **Your team** must have built images for all the TripInsights components and pushed them to the team's ACR. Share your understanding of how each of the images were built and pushed to the registry with your coach.

References

Docker

- Getting Started with Docker (https://docs.docker.com/get-started/)
- Docker Networking (https://docs.docker.com/v17.09/engine/userguide/networking)
- Dockerfile reference (https://docs.docker.com/engine/reference/builder/)
- Docker CLI reference (https://docs.docker.com/engine/reference/commandline/cli/)

SQL Server

- Getting Started with SQL Server Container (https://docs.microsoft.com/en-us/sql/linux/quickstart-install-connect-docker?view=sql-server-2017)
- Configuring SQL server (https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-docker)

Azure

- Azure CLI reference (https://docs.microsoft.com/en-us/cli/azure/get-started-with-azure-cli)
- Azure Container Registry (https://docs.microsoft.com/en-us/azure/container-registry/)
- Build with ACR (https://docs.microsoft.com/en-us/azure/container-registry/container-registry-tutorial-quick-task)

Trip Insights Source Code

- Containers OpenHack on GitHub (https://github.com/Microsoft-OpenHack/containers_artifacts)
- Testing POI API (https://github.com/Microsoft-OpenHack/containers_artifacts/tree/main/src/poi#testing)