Deploying a Flask App on Zoho Catalyst

This guide walks through deploying a standard Python Flask API (with two endpoints) on Zoho Catalyst. We'll cover setting up the Catalyst CLI, deciding between using Catalyst Functions (serverless) or AppSail (full app hosting), preparing the code accordingly, and deploying & testing. References to the official Zoho Catalyst docs are provided throughout.

1. Setting Up Catalyst CLI and Account

1. **Install the Catalyst CLI:** Download and install the Catalyst command-line tool. (For example, Catalyst publishes an npm package; ensure you have Node.js installed, then you can usually install with a command like npm install -g @zoho/catalyst-cli.) This CLI is how you interact with Catalyst from your terminal.
2. **Log in to your Catalyst account:** Run the command:

bash

CopyEdit

catalyst login

This will prompt you to allow error-reporting and to select a data center (e.g. US, EU, IN, AU, CA) based on your Catalyst accountdocs.catalyst.zoho.com. Your default browser will open to Zoho's login page; sign in and **Accept** the permissions. Once successful, you'll see a "login successful" message in your terminaldocs.catalyst.zoho.comdocs.catalyst.zoho.com.

3. **Initialize a new Catalyst project:** Choose (or create) a Catalyst organization and project for your app. In an empty folder for your project, run:

bash

CopyEdit

catalyst init

This will let you pick an organization and either select an existing project or create a new onedocs.catalyst.zoho.com docs.catalyst.zoho.com. (Note: your very *first* Catalyst project must be created via the web console – after that, you can create or import projects via the CLIdocs.catalyst.zoho.com.) Once initialized, the directory now becomes a Catalyst project with metadata (catalyst.json) and subfolders for any services you add.

2. Choosing Deployment Method: Functions vs AppSail

Catalyst supports two main ways to host your code:

- **Catalyst Functions (Serverless):** Write each API endpoint as an independent function. The CLI and Catalyst manage execution (similar to AWS Lambda). Use the **API Gateway** to expose HTTP routes. *Pros:* Auto-scaling, pay-per-use, easy to integrate with other Catalyst services. *Cons:* You must split your Flask logic into separate stateless functions, manage JSON events and routing, and use the Catalyst API Gateway for HTTP access. There are also limits (e.g. Catalyst currently supports Python 3.9 only for functions, and you must define each function's folder and dependencies).
- **Catalyst AppSail (PaaS):** Deploy your entire Flask app as a "service" on Catalyst. This is a fully managed PaaS for Java, Node.js, or Python web appsdocs.catalyst.zoho.com. You bundle your app (with a web server like Flask's) and Catalyst runs it on managed instances. *Pros:* Easier to port an existing Flask app with minimal changes. Supports full request/response frameworks. *Cons:* You manage an app process (though still serverless in that Catalyst auto-scales), and pricing may depend on number of instances. Only supported stacks (e.g. Python 3.9) are allowed.

For a **standard Flask app with two APIs**, AppSail is generally simpler: you can mostly keep your Flask code intact, and let Catalyst handle the web server. Catalyst even provides a Python SDK that supports Flask out of the box docs.catalyst.zoho.com. In fact, Zoho's docs say "AppSail: To host the independent Python Flask application. We will be building the front-end and back-end logic, and bundling them together to deploy on AppSail"docs.catalyst.zoho.com. Serverless Functions would require refactoring each route into a separate function and configuring API Gateway for routing.

Recommendation: Use **AppSail** unless you specifically need microservices or per-function billing. (If you did use Functions, keep in mind you would enable the Catalyst API Gateway and write tiny handler functions with signatures like def handler(context, event). Each would need its own folder and requirements, and you'd use catalyst functions:add to add themdocs.catalyst.zoho.comdocs.catalyst.zoho.com.)

3. Preparing the Flask App

If Using AppSail (PaaS)

1. **Install dependencies locally:** In your Flask app directory, install Flask and the Catalyst Python SDK **into your app folder**. For example:

bash

CopyEdit

```
python3 -m pip install --target . flask python3 -m pip install --pre --target . zcatalyst-sdk
```

This places the libraries in your project directory. (This matches Catalyst's example: it installs flask and zcatalyst-sdk into -t .docs.catalyst.zoho.com.)

2. **Initialize the AppSail service:** In the project directory (where your Flask app.py is), run:

bash

CopyEdit

```
catalyst appsail:init
```

Follow the prompts: choose **"N"** when asked to use a sample project (to use your own code), then confirm your current directory as the *source directory* (where your app code lives)docs.catalyst.zoho.com. Next select the *build path* (this is where Catalyst will assemble all files; it can be your source directory or a subdirectory). Then name your app (this is just an identifier, not changing your code). Finally pick **Stack: Python_3_9** and runtime **Python 3.9** (Catalyst only supports Python 3.9 for AppSaildocs.catalyst.zoho.com). After this, an app-config.json file is created in your source directory docs.catalyst.zoho.com, and your project's catalyst.json is updated to include the AppSail service.

3. **Configure the startup command:** By default, Catalyst needs to know how to launch your Flask app. For example, if your entry script is app.py, set the startup command. In the Catalyst console (or by editing app-config.json), use:

bash

CopyEdit

```
python3 -u app.py
```

This is exactly what the docs suggest as the *startup command* for Flaskdocs.catalyst.zoho.com. Also ensure your Flask code listens on the correct port: Catalyst sets the X_ZOHO_CATALYST_LISTEN_PORT environment variable (default 9000). In your app.py, use:

python

CopyEdit

```
import os if __name__ == '__main__': port = int(os.getenv('X_ZOHO_CATALYST_LISTEN_PORT', 9000))
app.run(host='0.0.0.0', port=port)
```

This snippet comes from Zoho's sample Flask appdocs.catalyst.zoho.com. It reads the port from the env or defaults to 9000.

4. **Include all files in the build:** Ensure that all your Python files and installed libraries are included in the build directory. Catalyst will zip up your build directory on deploydocs.catalyst.zoho.com. (You don't need to manually archive anything.) The docs note: *"Ensure all the Python application files along with the flask and zcatalyst-sdk modules are present in the build directory… Catalyst will automatically ZIP your app files during deployment"* docs.catalyst.zoho.com. In practice, if you ran pip install -t . above and your build path is the same folder, you're ready to go.

If Using Functions (Serverless)

If you opt for Catalyst Functions instead:

1. **Initialize a function:** Run catalyst functions:init (or catalyst init and select "Functions"). The CLI will ask for the function type and stack. Choose the appropriate stack (Python 3.9). Then give your function a folder name and an entry-point file (e.g. hello.py). The CLI creates a subdirectory (e.g. functions/hello/) with that .py file, a requirements.txt, and a catalyst-config.json. It also installs the Catalyst Python SDK and adds it to requirements.txt automaticallydocs.catalyst.zoho.com.

2. **Refactor routes into functions:** For each Flask route, write a corresponding function. For example, if you had @app.route('/hello'), create a function handler in hello.py:

python

CopyEdit

```
def handler(context, data): name = data.get('name', 'world') return {"message": f"Hello, {name}!"}
```

(Catalyst will call handler with the incoming JSON as data.) Each function must be standalone and stateless. Add any Python dependencies to requirements.txt in that folder. If you need multiple endpoints, repeat with separate functions or use one function to inspect data['path'] as needed. You can add more functions later with catalyst functions:add docs.catalyst.zoho.com.

3. **Setup API Gateway:** To expose your functions over HTTP, enable and configure the Catalyst API Gateway for your project. In the CLI run:

bash

CopyEdit

```
catalyst apigateway:enable
```

Then, create an api_gateway.json file describing the routes (method and URI) that map to each function. (Zoho's docs say: *"After you enable the API Gateway, you will be able to set up a JSON file… that contains the definitions of each API"* docs.catalyst.zoho.com.) For example, map GET /hello to your hello function. Once deployed, Catalyst will create actual endpoints you can call.

4. **Serve dependencies:** Each function folder can contain sub-modules and dependencies (per note in docs) docs.catalyst.zoho.com. Just be careful to include everything in requirements.txt. Also, if this is your first time

creating a Python function locally, set your Python path via catalyst config:set as the docs suggest docs.catalyst.zoho.com (CLI may prompt you).

4. Deploying and Consuming the APIs

1. **Deploy your code:** Once prepared, push your app to Catalyst. In your project directory, run:

bash

CopyEdit

```
catalyst deploy
```

This will deploy all initialized resources (functions, AppSail app, etc.) to the Catalyst console. You should see messages confirming upload of functions and AppSail. (Alternatively, you can deploy individually, e.g. catalyst functions:deploy or catalyst appsail:deploy.)

2. **Local testing (optional):** Before deploying, you can test your app locally. Catalyst CLI provides a local server:

bash

CopyEdit

```
catalyst serve
```

This launches a dev server (default port 9000) and serves your resources so you can hit your endpoints at http://localhost:9000/...docs.catalyst.zoho.com. This simulates the deployed environment (it even bundles the build path automaticallydocs.catalyst.zoho.com), letting you use tools like Postman locally.

3. **Access the live endpoints:** After a successful deploy, find your endpoints in the Catalyst console. For an AppSail service, Catalyst will display a service URL (e.g. https://<appname>-<org>.catalyst.zohoapps.com). You can invoke your Flask routes by appending paths (e.g. https://.../hello). For Functions behind the API Gateway, Catalyst provides public URLs for each defined route. You can now call these endpoints from anywhere (Postman, browser, curl, etc.).

4. **Testing externally:** Use a tool like Postman to send HTTP requests to your deployed endpoints. For example:
   - GET https://<your-app-url>/hello should return "Hello, World!" if you coded it that way.
   - POST with JSON {"name": "Alice"} to /hello (if implemented) and check the JSON response.

5. Pitfalls, Common Errors, and Limitations

- **First Project Must Use Console:** As noted, you cannot create your very first Catalyst project from the CLI; do that once in the web console, then use the CLI afterwarddocs.catalyst.zoho.com.
- **CLI Data Center Selection:** Always pick the correct data center at login. If your Catalyst account is in, say, the EU data center, choosing the wrong one will fail.
- **Python Path for Functions:** On first creating a local Python function, Catalyst may ask you to set your Python interpreter path via catalyst config:set. If you skip this, the function init can faildocs.catalyst.zoho.com.
- **Correct Directory Structure:** Make sure your source and build paths are set correctly. With AppSail, your *source directory* should contain your app code and libraries. With Functions, each function's folder must have its handler, requirements.txt, and catalyst-config.jsondocs.catalyst.zoho.com. Forgetting to include dependencies (or zcatalyst-sdk) will cause runtime errors.

- **Dependencies and Bundling:** For AppSail, install libraries into your project dir (not globally). The catalyst serve note points out that your specified build path is what gets served and zippeddocs.catalyst.zoho.com. If a library (like a C-extension) isn't compatible, you may need a different approach.
- **API Gateway JSON:** If using Functions, an improperly formatted api_gateway.json or forgetting to enable the gateway means your HTTP calls won't reach your functions. The CLI doc advises to carefully edit this JSON and use catalyst apigateway:enabledocs.catalyst.zoho.com.
- **Environment Variables:** In AppSail, use X_ZOHO_CATALYST_LISTEN_PORT for the port (default 9000) docs.catalyst.zoho.com. Hard-coding a different port in app.run() will break your app on Catalyst.
- **Debugging Functions:** You can use catalyst functions:shell to invoke and debug a function locally before deploy.

Limitations: Catalyst has some restrictions compared to AWS. It supports fewer runtimes (only Python 3.9, certain Node/Java versions). You have less control over infrastructure (since it's a managed PaaS) – you can't customize servers. Project quotas or service limits may apply (check Zoho's docs for any usage caps). In return, Catalyst integrates tightly with Zoho's ecosystem (e.g. Data Store, Authentication). Finally, remember that Catalyst is a newer platform: community resources are smaller than AWS's, and deployment/debugging relies on the Catalyst CLI and console.

Overall, for a simple Flask API, Catalyst AppSail provides a quick path: "zip & ship" your app with minimal changes and use catalyst deploy to host it. If you anticipate scaling to many microservices or need fine-grained function billing, then restructure your app into Catalyst Functions with API Gateway. Either way, follow the official steps and the pitfalls above to avoid common errors.

References: Official Zoho Catalyst docs and tutorials were used for all stepsdocs.catalyst.zoho.com docs.catalyst.zoho.comdocs.catalyst.zoho.comdocs.catalyst.zoho.comdocs.catalyst.zoho.comdocs.catalyst.zoho.com docs.catalyst.zoho.comdocs.catalyst.zoho.com. These show the CLI commands, file layouts, and procedures described.

Citations

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/login/login-from-cli/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/login/login-from-cli/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/initialize-resources/initialize-new-project/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/initialize-resources/initialize-new-project/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/initialize-resources/initialize-new-project/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/initialize-resources/initialize-appsail/

Catalyst Docs

https://docs.catalyst.zoho.com/en/serverless/help/appsail/help-guides/python/flask/

Catalyst Docs

https://docs.catalyst.zoho.com/en/tutorials/leadmanager-appsail/flask/introduction/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/initialize-resources/initialize-functions/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/initialize-resources/initialize-functions/

Catalyst Docs

https://docs.catalyst.zoho.com/en/serverless/help/appsail/help-guides/python/flask/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/initialize-resources/initialize-appsail/

Catalyst Docs

https://docs.catalyst.zoho.com/en/serverless/help/appsail/help-guides/python/flask/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/initialize-resources/initialize-appsail/

Catalyst Docs

https://docs.catalyst.zoho.com/en/serverless/help/appsail/help-guides/python/flask/

Catalyst Docs

https://docs.catalyst.zoho.com/en/serverless/help/appsail/help-guides/python/flask/

Catalyst Docs

https://docs.catalyst.zoho.com/en/serverless/help/appsail/help-guides/python/flask/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/initialize-resources/initialize-functions/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/working-with-api-gateway/introduction/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/serve-resources/serve-all-resources/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/serve-resources/serve-all-resources/

Catalyst Docs

https://docs.catalyst.zoho.com/en/cli/v1/login/login-from-cli/

All Sources

docs.catalyst.zoho