Tutorial: Develop your Python cloud applications offline with LocalStack

EuroPython 2023, July 18th, Prague

Waldemar Hummer
Thomas Rausch
Alexander Rashed

LocalStack

localstack.cloud

Agenda

- Intro to LocalStack
  - General background and overview
  - Basic installation and setup
- Developing Python Lambdas and Serverless applications locally
  - Serverless image resizer sample application
  - Lambda hot reloading
- Integration with Infrastructure-as-Code (IaC)
  - Developing an AppSync sample with CDK
  - Brief overview of other IaC frameworks (Terraform, Pulumi) with LocalStack
- Advanced Topics & Team Collaboration
  - State sharing via Cloud Pods
- Wrap up and outlook on future work

Intro

- Quick mini survey:
  - Do you have experience with AWS cloud?
    - → used it for hobby projects?
    - → use it in a professional setting on a daily basis?
  - Have you worked with LocalStack before?
  - Do you have experience with Infrastructure-as-Code tools? (Terraform, CDK, ...)

Some downloading in the background... :)

- Before we continue with a short intro...
- IF you have Docker installed locally, you can start pulling this image:

```
docker pull localstack/localstack-pro
```

```
pip install 'awscli-local[ver1]'
```

```
pip install 'awscli-local[ver1]'
```

Context:

Today's Cloud Development is Slow, Tedious, & Costly

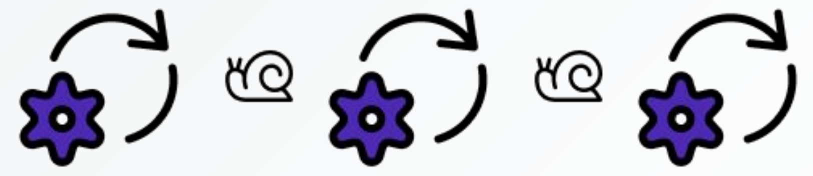
1

Alice is tasked with creating a new serverless Web application on AWS Cloud.

2

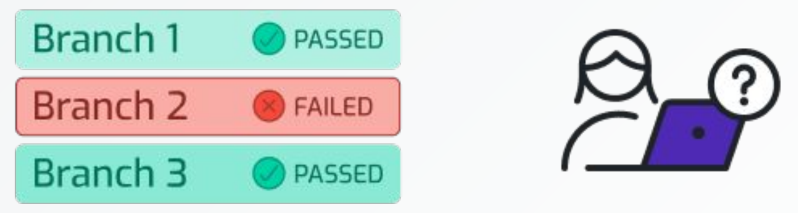
Developing on her local machine, she realizes that there are lots of dependencies with resources in the cloud (DBs, VMs, MQs, S3, ...)

3



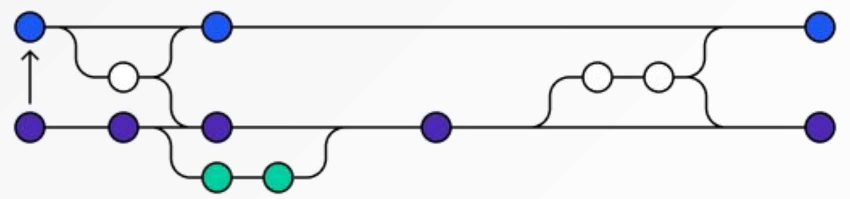
Alice realizes that the dev&test loop is extremely slow and tedious. Every local change needs to be packaged and uploaded to the cloud for testing.

4



Alice has a red build on her feature branch, but has troubles efficiently testing and debugging her code in the CI/CD pipeline.

5



Alice and her team are using Git flow for development - one CI build per feature branch. There is an explosion of different environments required for testing (branches * developers).

6

The dev manager approaches the team and complains that AWS test resources are not being cleaned up properly (causing a substantial cost spike in the last months).

What is LocalStack?

A fully functional local cloud stack - Develop your AWS cloud apps locally!

- Enables a highly efficient dev&test loop for cloud apps
- Ships as a Docker image, easy to install and start up
- Support for ~70 AWS APIs (and growing):
  - compute (e.g., Lambda, ECS, EKS)
  - various databases (e.g., DynamoDB, RDS, DocumentDB)
  - messaging (e.g., SQS, Kinesis, MSK)
  - some sophisticated/exotic APIs (e.g., QLDB, Athena, Glue)
- Advanced collaboration features and CI integrations
  - redefining the way cloud apps are developed across the lifecycle!

Our Story

2016 August	First open source commits of LocalStack in GitHub repository
2018 March	Reaching 10.000 stars on GitHub
2019 July	LocalStack Pro extensions first version available
2020 July	Reaching 25.000 stars on GitHub, 50+ AWS services supported in LocalStack
2022 May	Reached 20+ team members, and growing!
2022 July	Release of LocalStack version 1.0.0
2023 March	Release of LocalStack version 2.0.0

47.000+
GITHUB STARS

150.000.000+
DOCKER PULLS

20.000+
SLACK MEMBERS

460+
CONTRIBUTORS

Installation & Getting Started

Configuring the API Key

- LocalStack provides different product tiers
  - Open Source (Community) version
  - LocalStack Pro / Team / Enterprise

- Some features in the demo will require advanced Pro/Team features
  - → we'll configure an API key to enable these features

- We have prepared a special trial API key for this tutorial: **ls-ep-23-demo**
  - Please make sure to configure this in **all terminal sessions**:

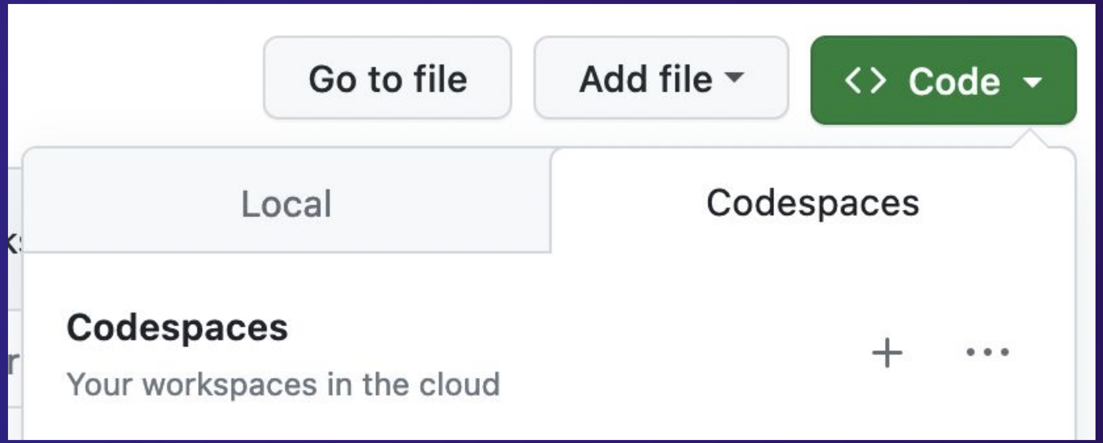
```
export LOCALSTACK_API_KEY=ls-ep-23-demo
```

Two Installation Options

- Repository with code samples:
  - https://github.com/localstack/localstack-workshop

- **Option 1**: Install LocalStack on the local machine (requires Docker)
  - Clone project onto your local machine
  - Install the CLI via pip:
    ```
    pip install --pre localstack
    ```
  - Start up LocalStack:
    ```
    localstack start
    ```

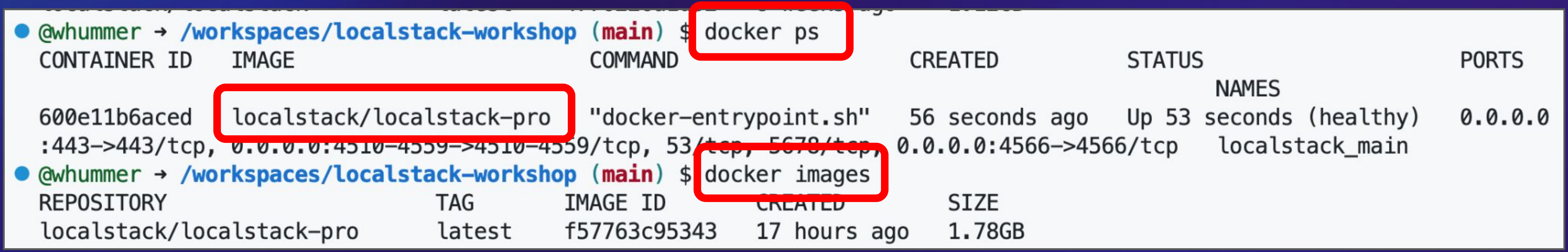
- **Option 2**: Use an online IDE (Github Codespace) to follow the tutorial
  - Open the repo link on Github
  - Launch the project in a Github Codespaces environment



Codespaces
Your workspaces in the cloud

Installation Result

- After successful installation and startup, you should see something like this:



```
● @whummer → /workspaces/localstack-workshop (main) $ docker ps
  CONTAINER ID   IMAGE                        COMMAND                  CREATED          STATUS                     PORTS
                                                                                                      NAMES
  600e11b6aced   localstack/localstack-pro    "docker-entrypoint.sh"   56 seconds ago   Up 53 seconds (healthy)   0.0.0.0
  :443->443/tcp, 0.0.0.0:4510-4559->4510-4559/tcp, 53/tcp, 5678/tcp, 0.0.0.0:4566->4566/tcp   localstack_main
● @whummer → /workspaces/localstack-workshop (main) $ docker images
  REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
  localstack/localstack-pro   latest    f57763c95343   17 hours ago     1.78GB
```

- Common issues and resolutions:
  - If you get a startup error related to port 53, try: `export DNS_ADDRESS=0`
  - Make sure the **localstack/localstack-pro** image is used (otherwise check API key)

Configuration Options

- LocalStack is a highly configurable platform
  - **Network** configs (e.g., ports, Docker network, …)
  - **Service-specific** configurations (e.g., Kinesis latency, ECS Docker flags, …)
  - **Lambda** configs (e.g., local code mounting, executor mode, …)
  - **Debug** settings (e.g., verbose logs, debug ports)
  - **Persistence** settings (e.g., data directory, persistence mode, …)
  - …
- Sensible defaults used for most configs, based on the user's environment
  - Utilities to validate the config: "`localstack config validate`" CLI

- More details in our online docs: https://docs.localstack.cloud

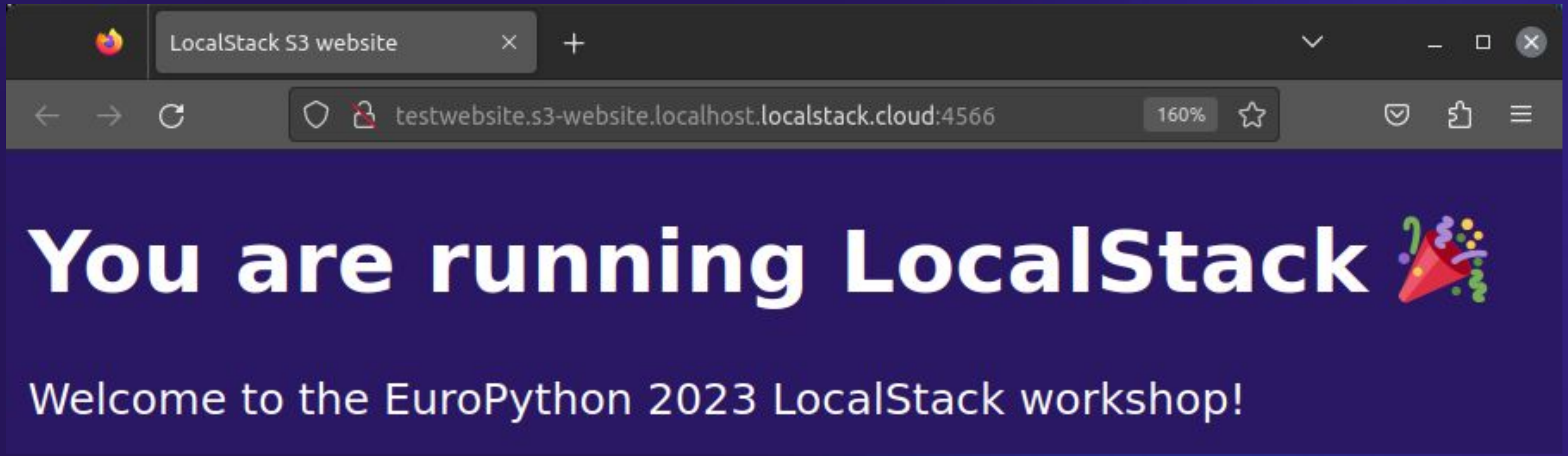
Getting Started:
S3 Website - Hello World!

Hello World - S3 Website App

- See `depoy.sh` script in the `hello-world/` folder in the repo

```
awslocal s3api create-bucket --bucket testwebsite
awslocal s3api put-bucket-policy --bucket testwebsite --policy file://bucket-policy.json
awslocal s3 sync ./ s3://testwebsite
awslocal s3 website s3://testwebsite/ --index-document index.html
```

- Once deployed, we can open the S3 website in the browser
  - http://testwebsite.s3-website.localhost.localstack.cloud:4566



You are running LocalStack 🎉

Welcome to the EuroPython 2023 LocalStack workshop!

Web Application

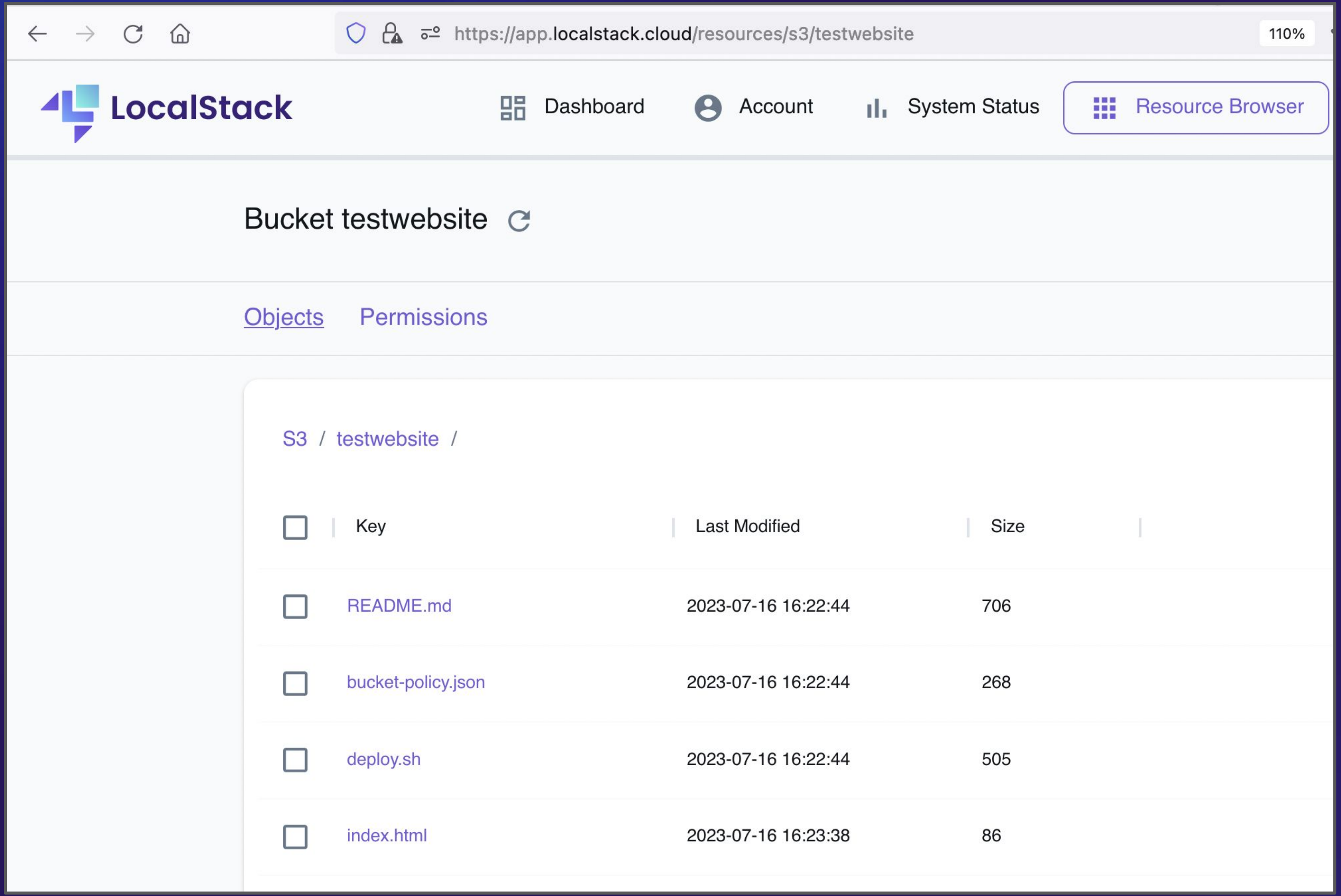
- LocalStack offers a graphical Web user interface
  - Usage dashboard, account management, …
  - **Resource Browser** - simplified version of the AWS Console locally
  - https://app.localstack.cloud

- Feel free to sign up for a user account directly!

- Alternatively, we've prepared a demo account for this tutorial:
  - Username: `demo@localstack.cloud`
  - Password: `EuroPython23!`

Web App - Resource Browser

- We can use the Resource Browser to inspect the files in our local S3 bucket:



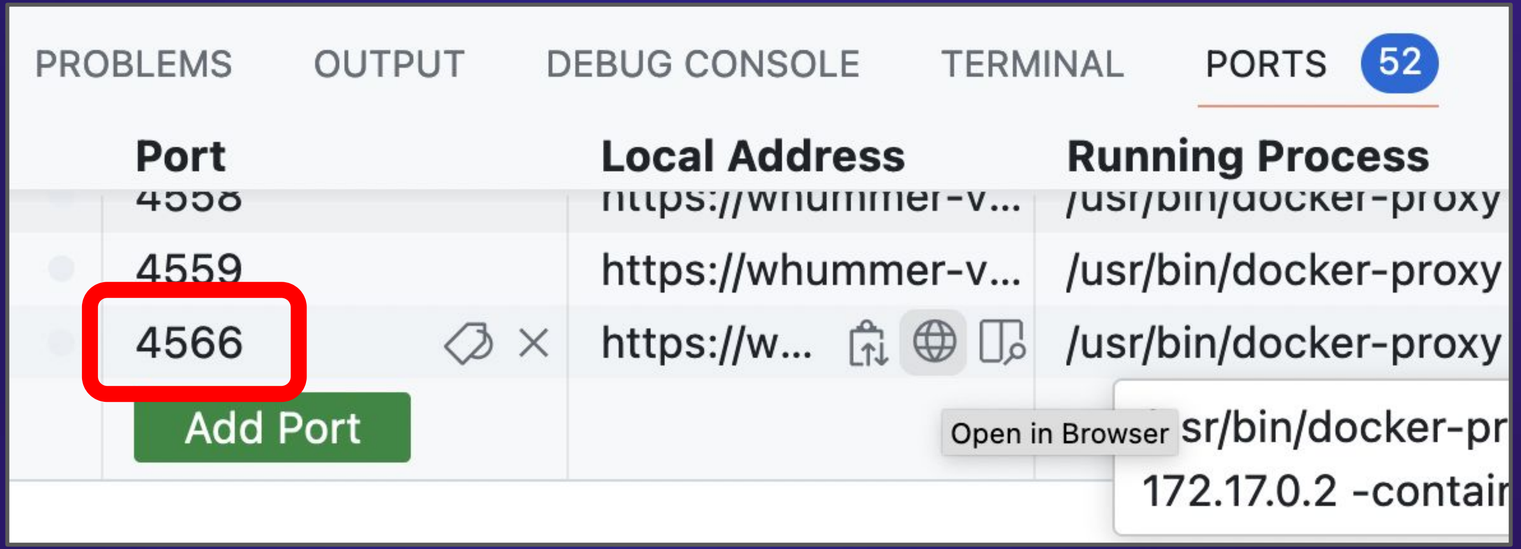
https://app.localstack.cloud/resources/s3/testwebsite

Bucket testwebsite

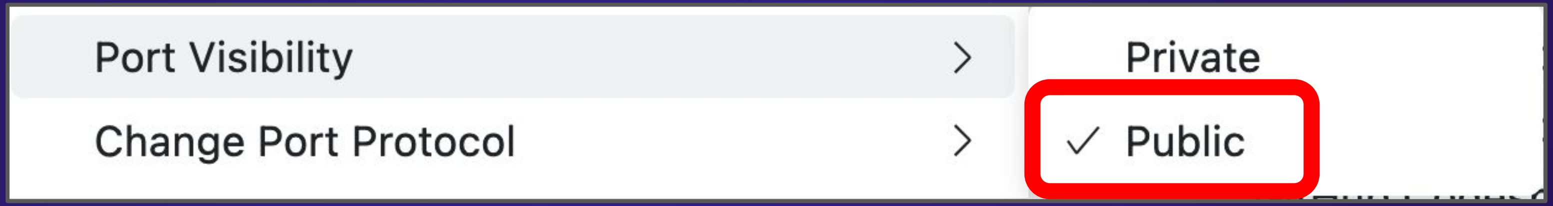
S3 / testwebsite /

Key	Last Modified	Size
README.md	2023-07-16 16:22:44	706
bucket-policy.json	2023-07-16 16:22:44	268
deploy.sh	2023-07-16 16:22:44	505
index.html	2023-07-16 16:23:38	86

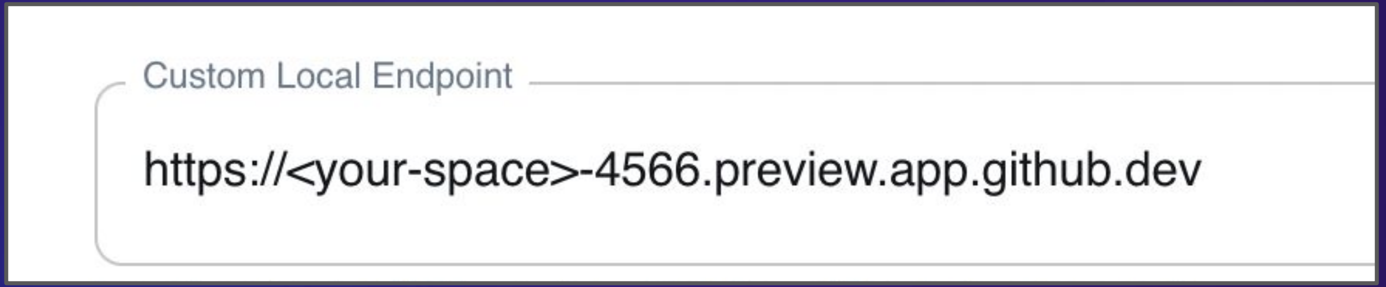
- Note: In Codespaces we need to configure the Web app to use the custom endpoint URL to connect to:



Port	Local Address	Running Process
4558	https://whummer-v...	/usr/bin/docker-proxy
4559	https://whummer-v...	/usr/bin/docker-proxy
4566	https://w...	/usr/bin/docker-proxy



Port Visibility: Private [✓ Public]



Custom Local Endpoint

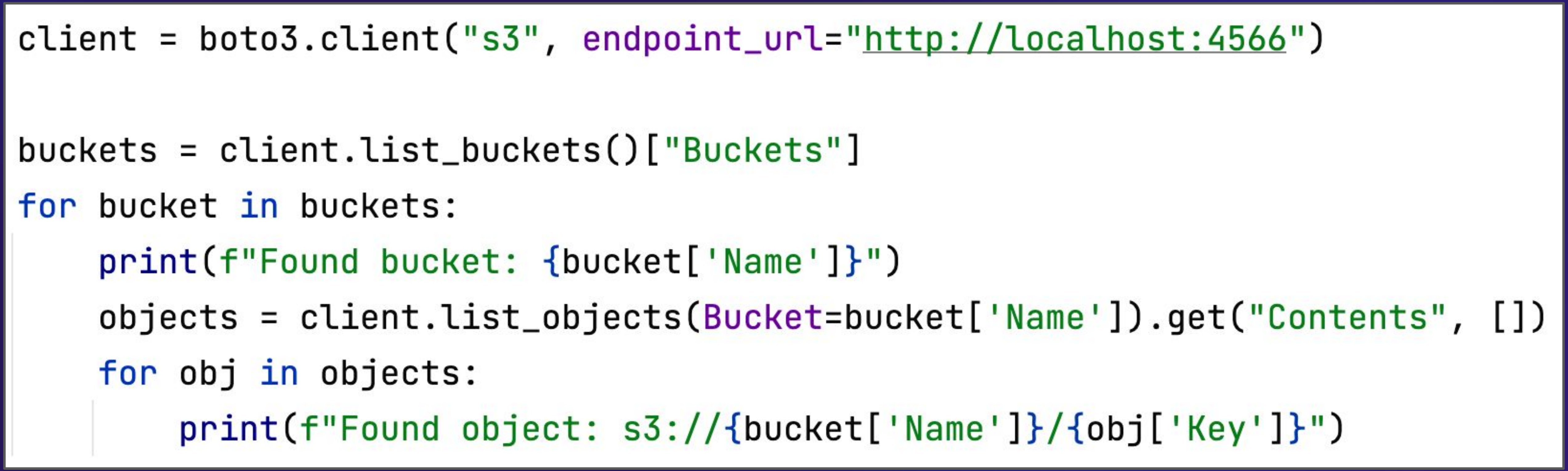
https://<your-space>-4566.preview.app.github.dev

Connecting to AWS from Python

- AWS provides SDKs for various programming languages (Java, Go, .NET, Node.js, …)

- For Python, the official AWS SDK is called **`boto3`** (https://github.com/boto/boto3)
  - → we will see this quite frequently in our sample apps

- Simple example of connecting to S3 - listing all buckets and objects:



```python
client = boto3.client("s3", endpoint_url="http://localhost:4566")

buckets = client.list_buckets()["Buckets"]
for bucket in buckets:
    print(f"Found bucket: {bucket['Name']}")
    objects = client.list_objects(Bucket=bucket['Name']).get("Contents", [])
    for obj in objects:
        print(f"Found object: s3://{bucket['Name']}/{obj['Key']}")
```

run `python list_s3_objects.py` in the repo

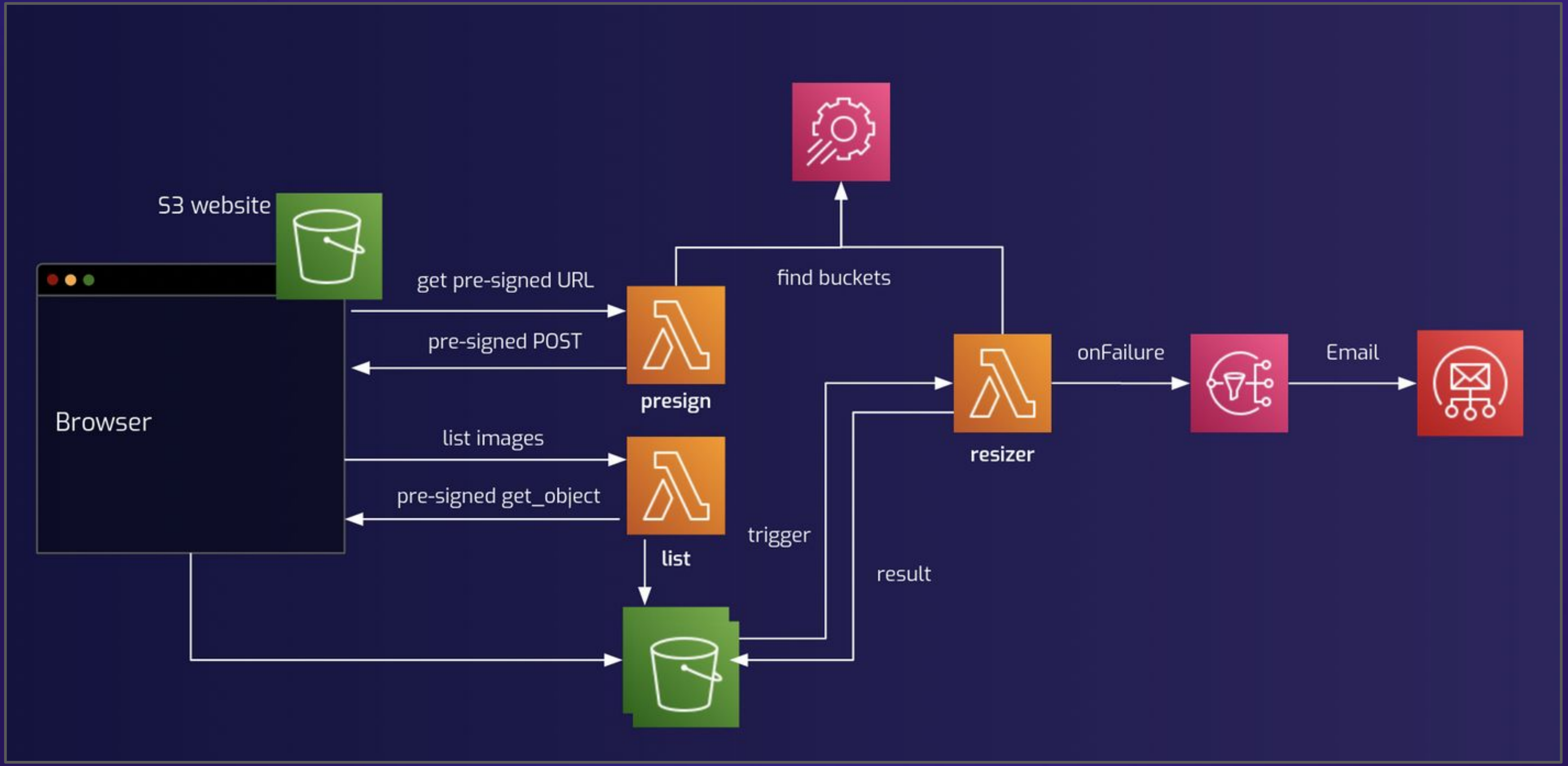
Sample 1:
Serverless Image Resizer

Application Architecture

- Allows to resize images
  - Upload image files to S3
  - Triggers a Lambda
  - Result is put back onto S3

- Covers several AWS services
  - Lambda functions
  - S3 bucket
  - SSM parameter
  - SNS failure notification

- Ships with a simple Web UI



Installation

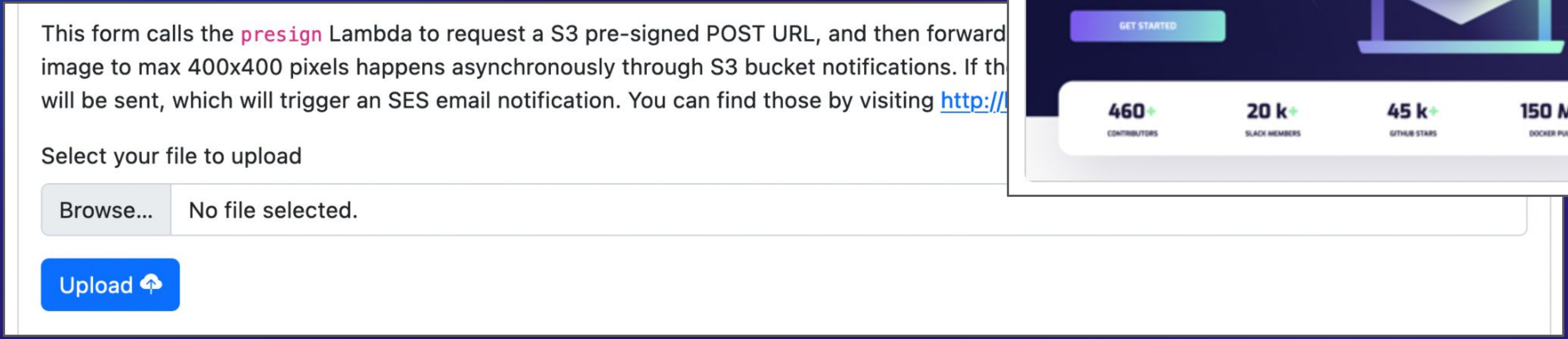
- To install the sample application, we can use the **`bin/deploy.sh`** script in **`01-serverless-image-resizer/`**

- The installation script performs the following steps:
  - Creates **basic infrastructure**: the S3 bucket, SSM parameters, SNS topics
  - Creates the **Lambda functions** for (1) list, (2) presign, and (3) resizing
    - Note: the resizer Lambda depends on some image modification libraries (pillow) - for MacOS the build is executed in a Docker container, to avoid compile conflicts
  - Creates an API Gateway to expose the Lambdas via an HTTP endpoint

- Finally, the script uploads the HTML/JavaScript sources for the WebUI
  - → see next slide on how to use the UI

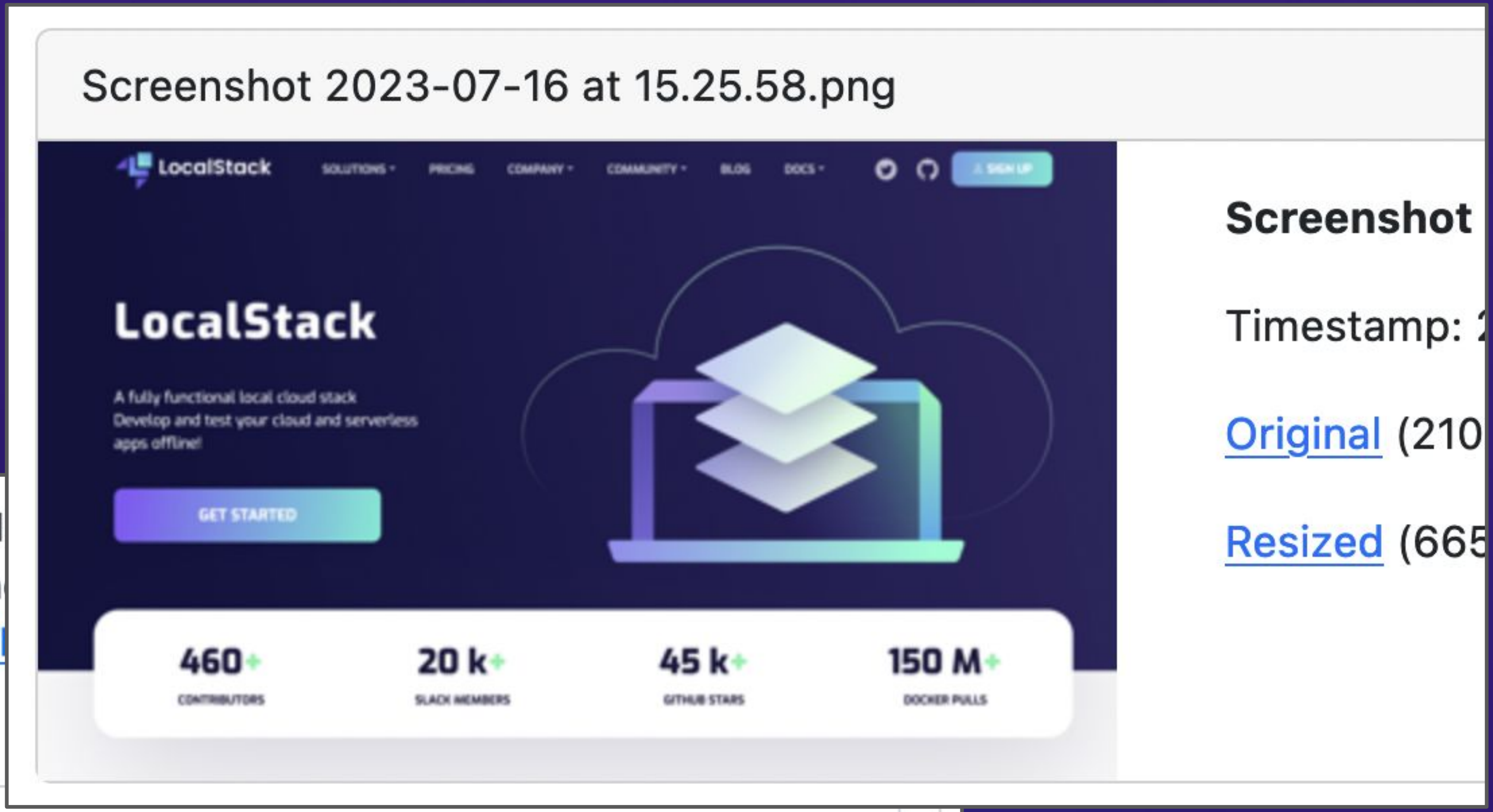
Resizer Web UI

- Web UI to upload and resize images



This form calls the presign Lambda to request a S3 pre-signed POST URL, and then forward
image to max 400x400 pixels happens asynchronously through S3 bucket notifications. If th
will be sent, which will trigger an SES email notification. You can find those by visiting http://

Select your file to upload



Screenshot 2023-07-16 at 15.25.58.png

Screenshot

Timestamp:

Original (210

Resized (665

- Once the app is deployed, the Web UI is accessible here:
  - Locally: http://webapp.s3-website.localhost.localstack.cloud:4566
  - Codespace: https://<your-space>-4566.preview.app.github.dev/webapp/index.html

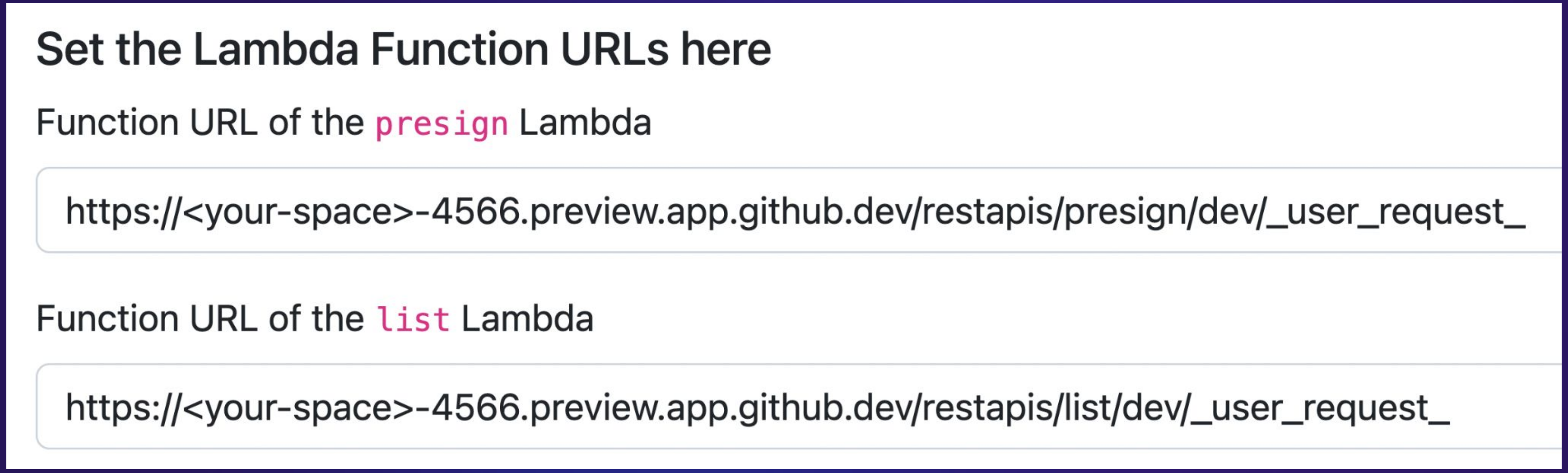
Resizer Web UI - Configuration

- The Web UI needs to be configured with the .
- Copy the output from these commands:

```
awslocal lambda list-function-url-configs --function-name presign
```

```
awslocal lambda list-function-url-configs --function-name list
```

- Note: For Codespaces, we need to configure slightly different endpoints:
  - `<your-space-port-4566>/restapis/presign/dev/_user_request_`
  - `<your-space-port-4566>/restapis/list/dev/_user_request_`



Set the Lambda Function URLs here

Function URL of the `presign` Lambda

https://<your-space>-4566.preview.app.github.dev/restapis/presign/dev/_user_request_

Function URL of the `list` Lambda

https://<your-space>-4566.preview.app.github.dev/restapis/list/dev/_user_request_

Lambda Hot Reloading

- What if we want to make changes to the existing functionality?
  - With **hot reloading**, changes are immediately reflected (→ fast feedback cycles!)

- We'll now extend the example and modify the "list" Lambda function:
  - See the hint in the `bin/deploy.sh` script

```
awslocal lambda update-function-code --function-name list \
  --s3-bucket hot-reload --s3-key "$(pwd)/lambdas/list"
```

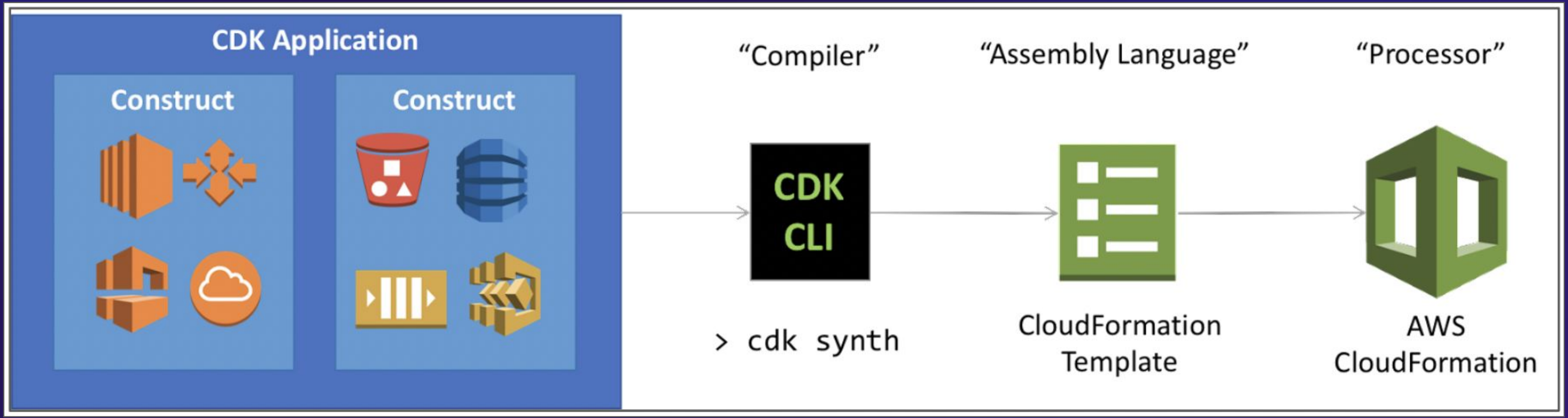
special bucket name   absolute path to Lambda code

- We can now modify the Lambda handler → the result is immediately reflected in the UI
  - → see notes in the file `lambdas/list/handler.py`

Sample 3:
AppSync GraphQL API with CDK

AWS Cloud Development Kit (CDK)

- AWS CDK is a popular tool to define your Infrastructure as Code (IaC)
- Offers bindings for various programming languages
  - Python, TypeScript, Go, Java, …
- On deployment, CDK programs are compiled into CloudFormation stacks



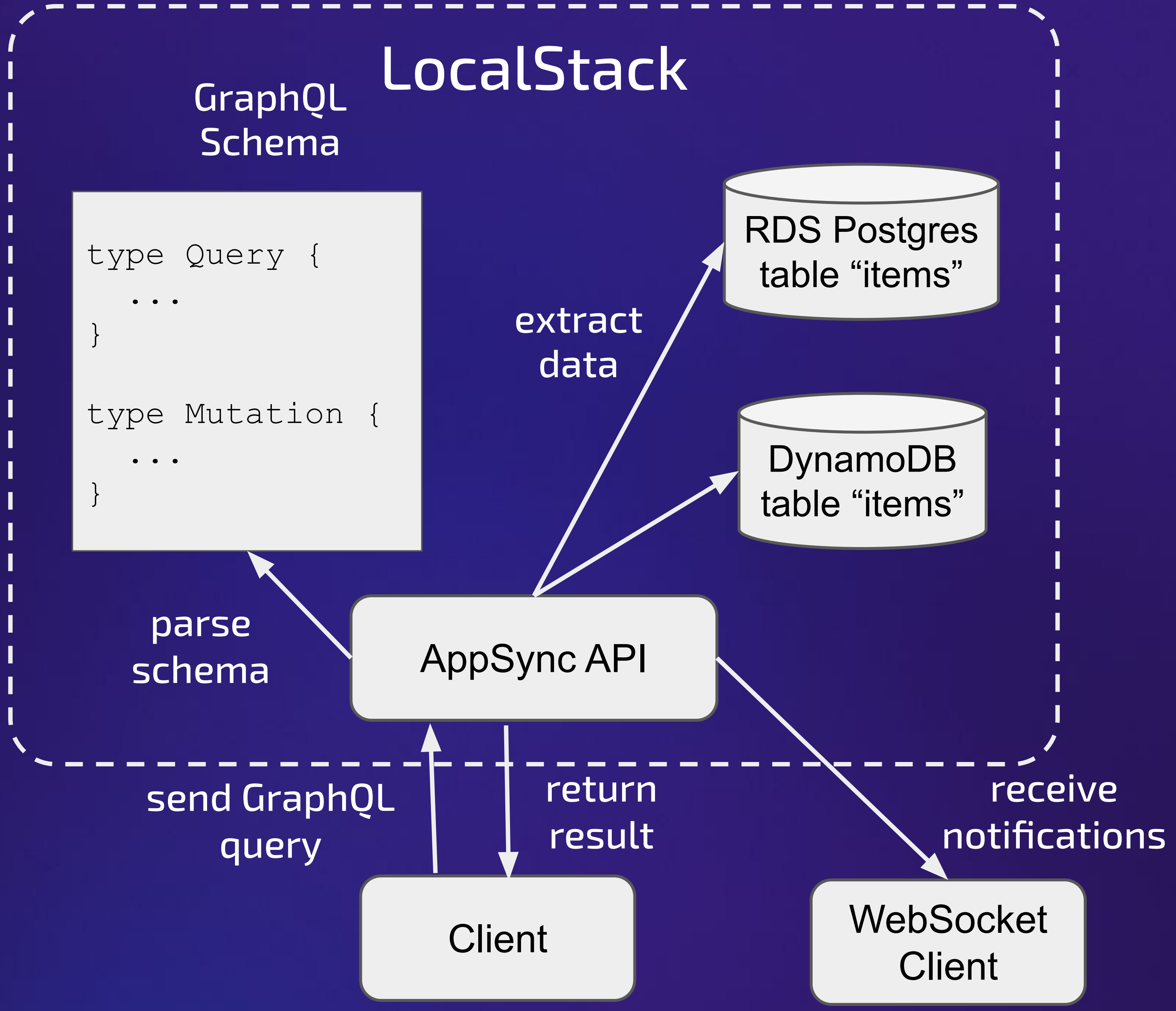
- The CLI tool `cdklocal` simplifies deploying CDK stacks against LocalStack
  - https://github.com/localstack/aws-cdk-local

Sample Application - AppSync GraphQL

- CDK script defines resources in Python

- Creates an AppSync application:
    - GraphQL schema
    - DynamoDB table
    - RDS database
    - Resolver mappings
    - ...

- Run a GraphQL **mutation** to add items
- Run a GraphQL **query** to retrieve items



```
type Query {
  ...
}

type Mutation {
  ...
}
```

CDK Application

- CDK script defines resources in a declarative way

- We use **`cdklocal`** for deployment:

```
pip install aws-cdk-local
```

- Deploy the resources (from **`cdk`** dir):

```
cdklocal bootstrap
```

```
cdklocal deploy
```

(There's also a **`make`** target, for convenience:)

```
make deploy
```

```python
from aws_cdk import ...

class CdkStack(Stack):
  def __init__(self, scope: Construct, id: str, **kwargs):
    super().__init__(scope, id, **kwargs)

    # RDS Serverless Postgres DB cluster
    cluster = rds.ServerlessCluster(self, "Database",
        default_database_name="testappsync", ...
    )
    # DynamoDB table
    table = dynamodb.Table(self, "Table",
        table_name="table1",
        partition_key=dynamodb.Attribute(
            name="id", type=dynamodb.AttributeType.STRING)
    )
    # GraphQL API
    api = appsync.GraphqlApi(self, "GraphqlApi",
        name="test-api",
        schema=appsync.SchemaFile.from_asset("schema.graphql")
    )
    ...
```

Interacting with the application

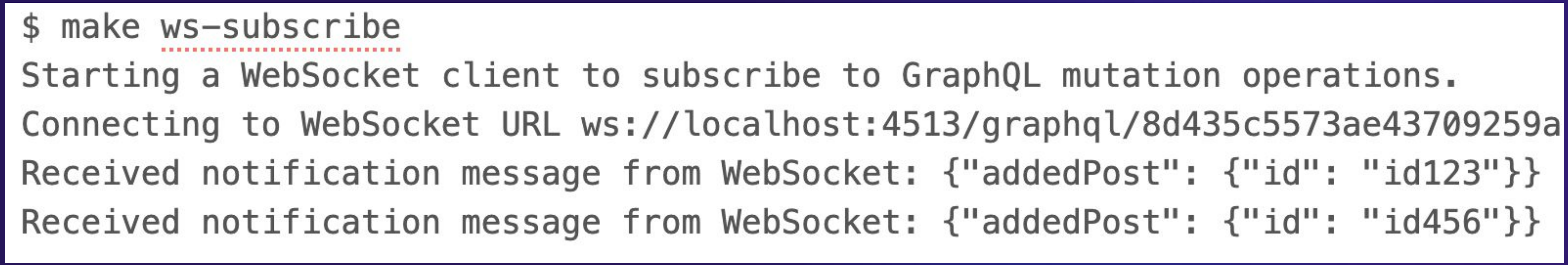
- Once deployed, we can use an HTTP client to interact with the GraphQL API
  - → see **run.sh** script in the repository
- Adding items to the DynamoDB (or RDS) table:

```
curl -d '{"query":"mutation {addPostDDB(id: \"id123\"){id}}"}' \
    -H "x-api-key: …" … http://localhost:4566/graphql/$api_id
```

- Listing items from the DynamoDB (or RDS) table:

```
curl -d '{"query":"query {getPostsDDB {id}}"}' \
    -H "x-api-key: …" … http://localhost:4566/graphql/$api_id
```

- We can start a WebSocket subscriber in a separate terminal to receive notifications:



```
$ make ws-subscribe
Starting a WebSocket client to subscribe to GraphQL mutation operations.
Connecting to WebSocket URL ws://localhost:4513/graphql/8d435c5573ae43709259a
Received notification message from WebSocket: {"addedPost": {"id": "id123"}}
Received notification message from WebSocket: {"addedPost": {"id": "id456"}}
```

Misc. Integrations & LocalStack in CI Pipelines

Growing Ecosystem of Integrations

CI/CD Systems

GitLab
circleci
GitHub Actions
HARBOR
Shipyard

IaC Tools

HashiCorp Terraform
Pulumi
aws Cloud Development Kit

Programming Language SDKs

python™
spring
JUnit 5
.NET

App Development Frameworks

Testcontainers
namespace
serverless
AWS SAM

Local Dev Tools

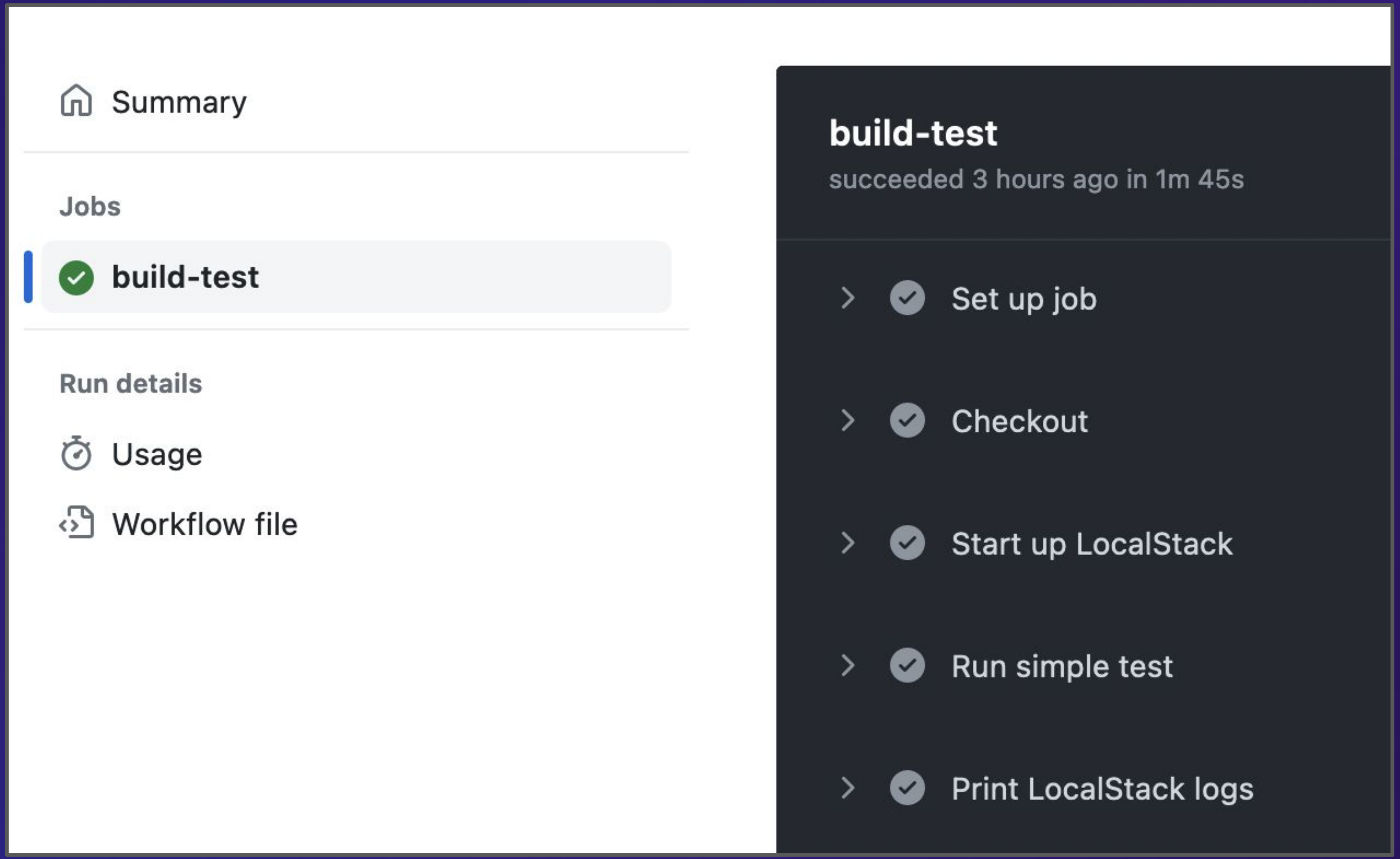
docker Compose
AWS CLI
AWS CloudFormation

Taking a look at our CI configuration …

- The Workshop project comes with a simple Github Actions CI
  - See configuration file under `.github/workflows/build-test.yml`

- Easy to make the switch and promote changes
  - Local → CI/CD → staging → production

- Using LocalStack in CI has several benefits:
  - Fosters repeatable builds, frequent testing
  - Full isolation: no resource naming conflicts
  - Quicker turnaround times (10x faster builds)
  - Less costs as compared to real cloud resources
  - …



build-test
succeeded 3 hours ago in 1m 45s

- Set up job
- Checkout
- Start up LocalStack
- Run simple test
- Print LocalStack logs

Integration with other IaC Frameworks

- LocalStack natively supports a number of other IaC and application dev. frameworks

- Terraform
  - `tflocal` command-line: https://github.com/localstack/terraform-local

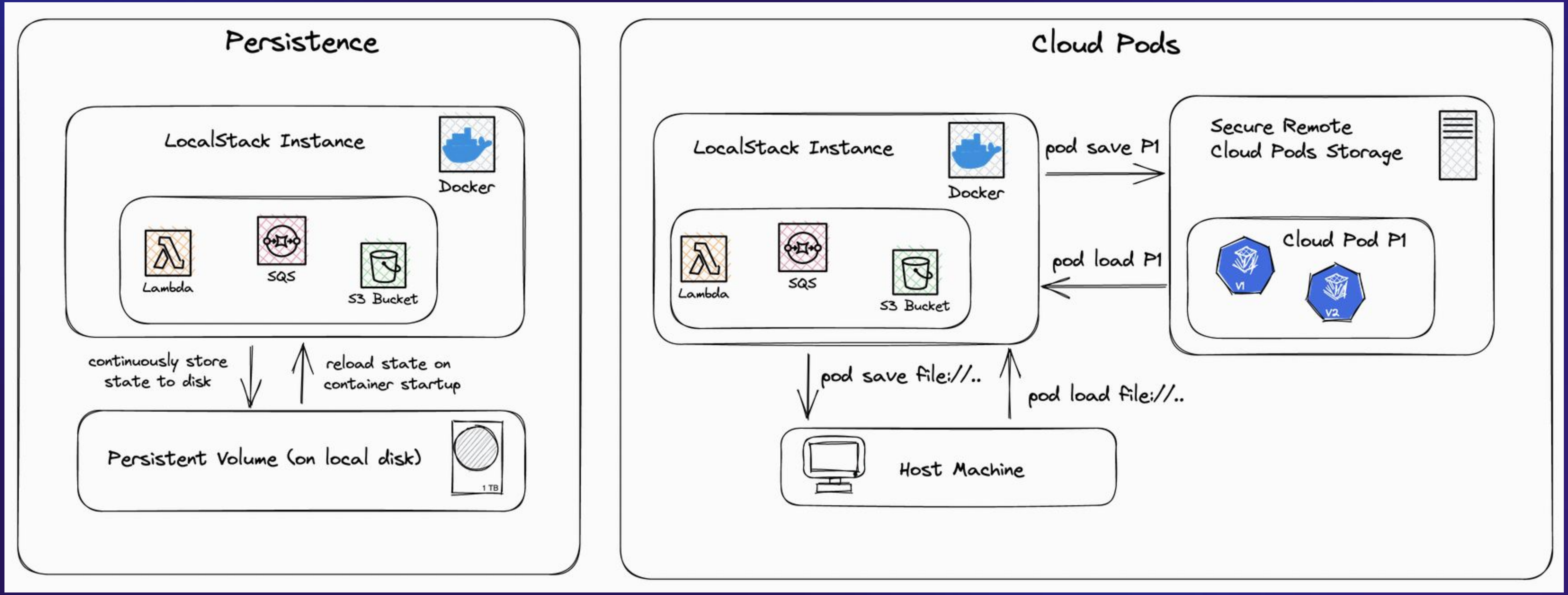
- Pulumi
  - `pulumilocal` command-line: https://github.com/localstack/pulumi-local

- Serverless.com Framework
  - `serverless-localstack` plugin: https://github.com/localstack/serverless-localstack

Sample 4:
Cloud Pods & Persistence

State Sharing via Cloud Pods

- By default, the state in LocalStack is **ephemeral**, i.e., not persisted across restarts
- **Cloud Pods** allow to take a **persistent snapshot** of the LocalStack application state
  - → push it to a server, share it with team members, load it back into the instance, …



Cloud Pods can be stored locally or remotely

- First, let's create some resources - S3 buckets, SQS queues, …:

```
awslocal s3 mb s3://test-bucket
awslocal sqs create-queue --queue-name q1
awslocal cognito-idp create-user-pool --pool-name p1
```

- We can then store the state to a local cloud pod file:

```
localstack pod save file://my-cloud-pod.zip
```

- Now - let's restart the LocalStack instance, and load the state back in:

```
localstack pod load file://my-cloud-pod.zip
```

Remote: Let's store and share some state!

- First, create an S3 bucket and add some files to it:

```
awslocal s3 mb s3://test-bucket
awslocal s3 cp local-file.png s3://test-bucket
```

- Make sure your CLI and Web app are signed in using:

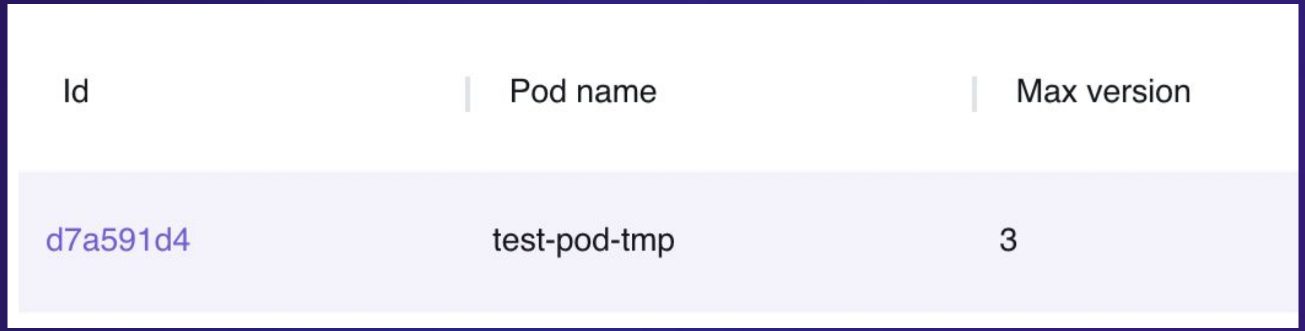
```
demo@localstack.cloud
```

```
EuroPython23!
```

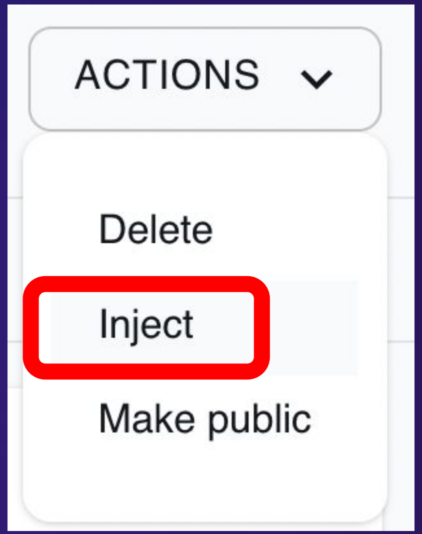
- Choose a sufficiently random name, then save a cloud pod:

```
localstack pod save my-pod-512398
```

- Browse the existing cloud pods in the Web application, then inject one!
    - https://app.localstack.cloud/pods
    - This way you can see the resources and files from your colleagues!



Id	Pod name	Max version
d7a591d4	test-pod-tmp	3



Wrap Up

Summary

- In this tutorial, we've learned how to:
  - Install & configure LocalStack
    - On your your local machine, in Codespaces, and in your CI pipeline
  - How to develop and test AWS serverless applications fully locally
    - → Lambda hot reloading helps speed up the local feedback cycles
  - How to seamlessly integrate IaC frameworks across local & CI/CD
    - Repeatability is key - spinning up sandbox environments to **foster frequent testing**

- We've peeked into some advanced features
  - Cloud Pods for state sharing - opens up new possibilities for team collaboration
  - LocalStack integrations - various tools across the SDLC

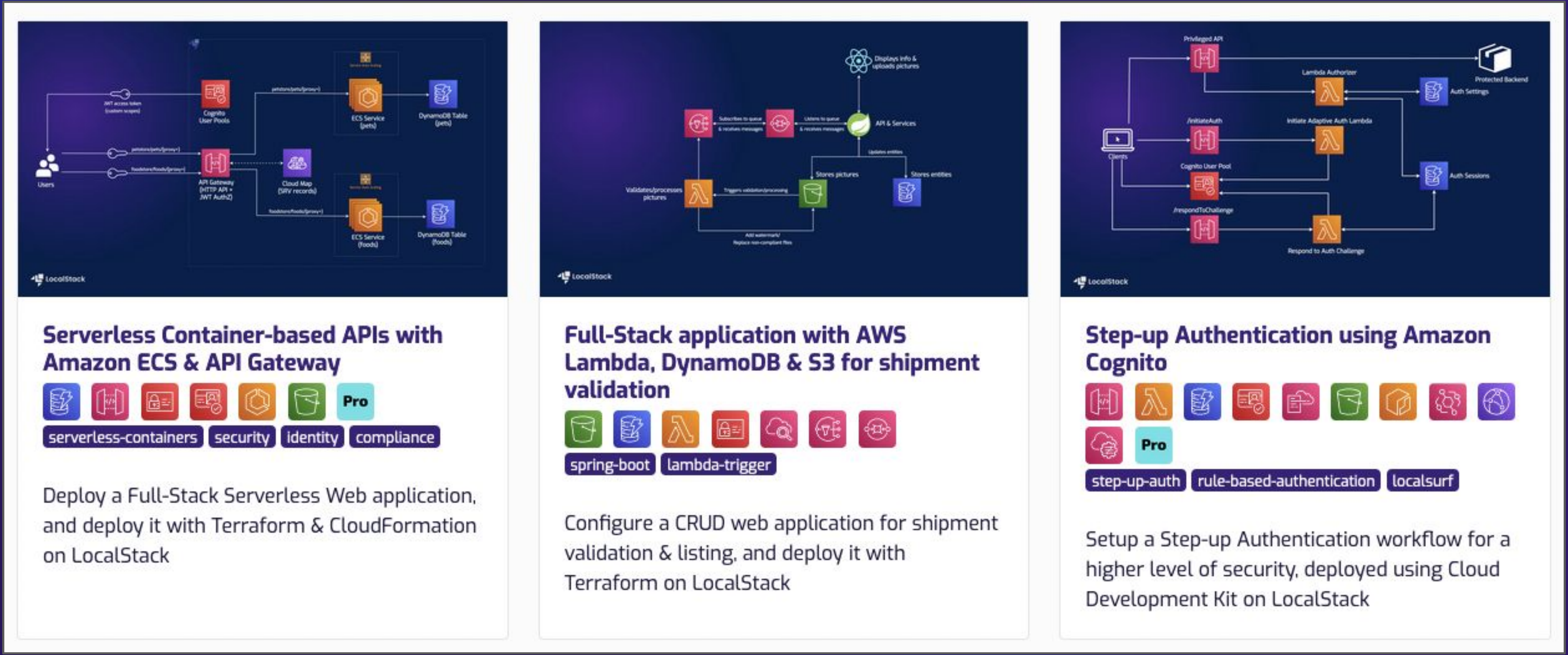
Exciting Features in the Pipeline

- **LocalStack Extensions**
  - Mechanism to plug in additional functionality into LocalStack
  - Several core use cases: fault injection, time simulation, hybrid setups (AWS proxy), etc
- **IAM Live Policy Generation**
  - IAM can be a major pain point when working with AWS
  - We're working on fully automated policy generation (based on observed user requests)
- **Ephemeral LocalStack Cloud Sandboxes**
  - "1-click deploy" of a LocalStack instance that runs in a temporary remote instance
  - → can be used to create preview environments of applications
- **Deeper integration of Cloud Pods & Stack Insights with CI builds**
  - → providing insights and facilitate debugging

… and much more!

Docs & Developer Hub

- **Docs:** Detailed list of services, config options, service coverage, etc
- **Developer Hub:** Curated list of sample applications
  - https://docs.localstack.cloud/applications



Serverless Container-based APIs with Amazon ECS & API Gateway

Pro

serverless-containers | security | identity | compliance

Deploy a Full-Stack Serverless Web application, and deploy it with Terraform & CloudFormation on LocalStack

Full-Stack application with AWS Lambda, DynamoDB & S3 for shipment validation

spring-boot | lambda-trigger

Configure a CRUD web application for shipment validation & listing, and deploy it with Terraform on LocalStack

Step-up Authentication using Amazon Cognito

Pro

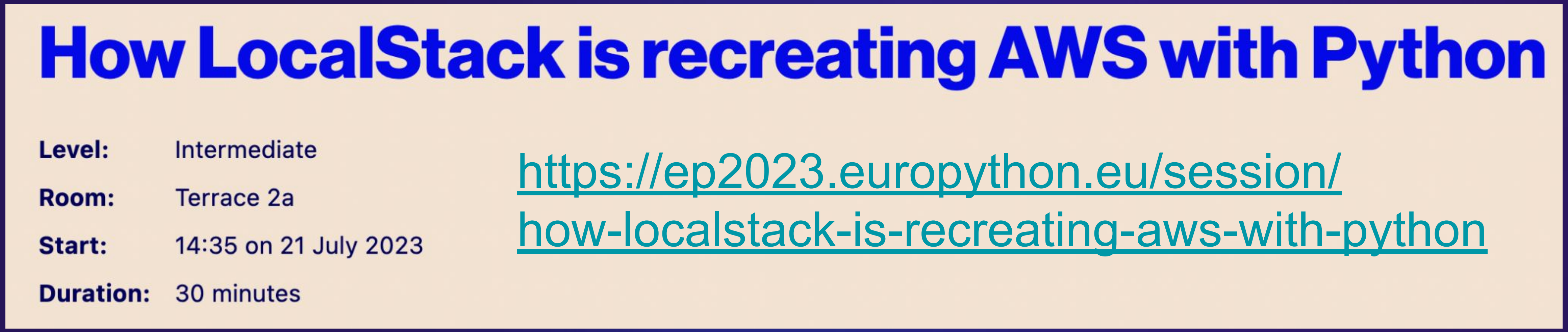
step-up-auth | rule-based-authentication | localsurf

Setup a Step-up Authentication workflow for a higher level of security, deployed using Cloud Development Kit on LocalStack

A Little Promo

- **Use the QR Code to get 3 months of LocalStack Pro for free!**
  - Also: Please reach out to us if you'd like a deep-dive demo for your team 💻
- **We'd love to have you in our community:**
  - Slack Community: https://slack.localstack.cloud
  - Monthly Meetups (online): https://meetup.com/localstack-community
  - Discuss Forum: https://discuss.localstack.cloud
  - Github: https://github.com/localstack/localstack

- **Check out this talk to learn about the internals of LS!! →**



How LocalStack is recreating AWS with Python

Level:	Intermediate
Room:	Terrace 2a
Start:	14:35 on 21 July 2023
Duration:	30 minutes

https://ep2023.europython.eu/session/how-localstack-is-recreating-aws-with-python

Backup

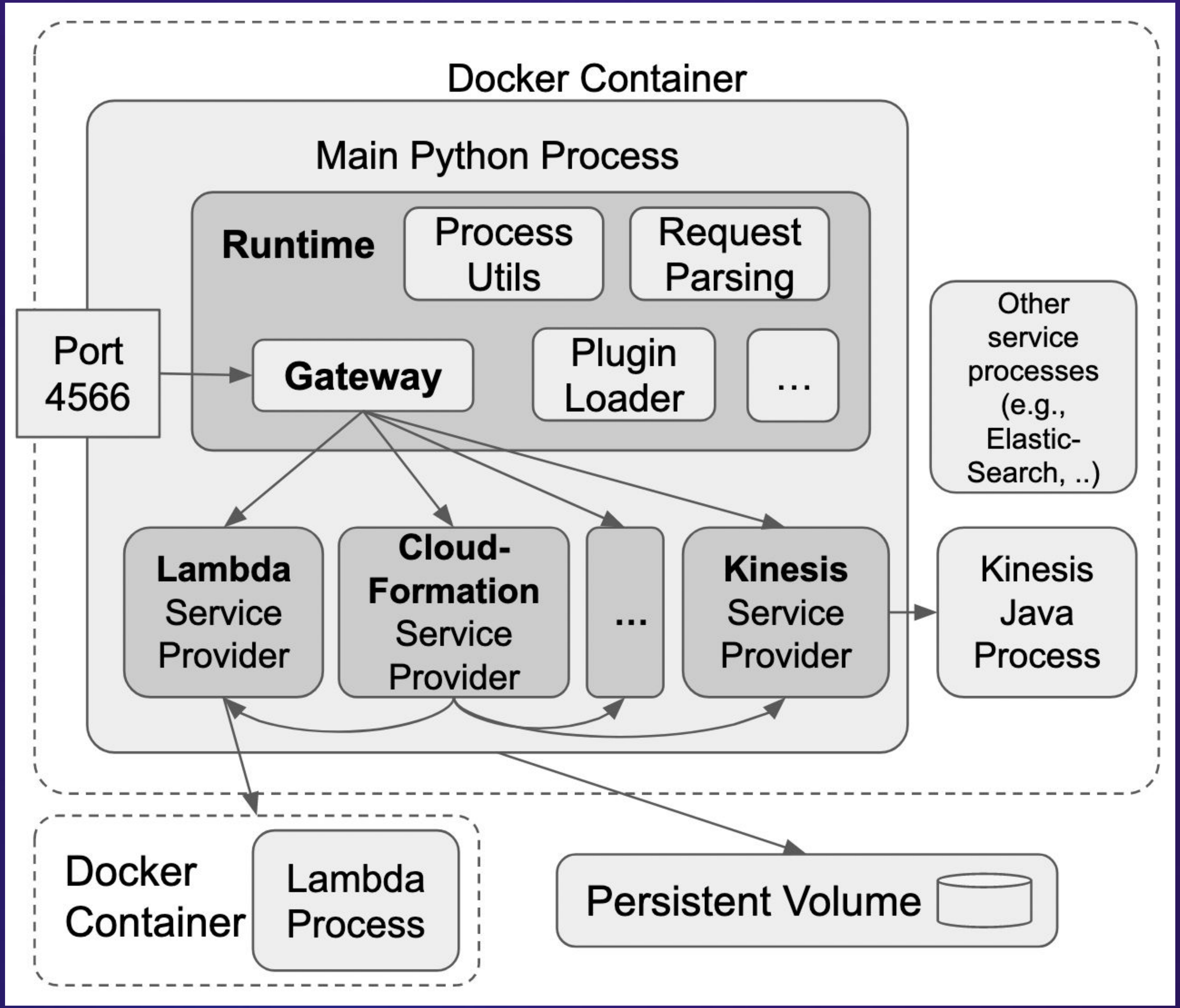
Sample 4:
Athena SQL queries over S3 files

Athena SQL Queries over S3

- Following the steps in this sample application:
  - https://github.com/localstack-samples/sample-query-data-s3-athena-glue

LocalStack Architecture Overview

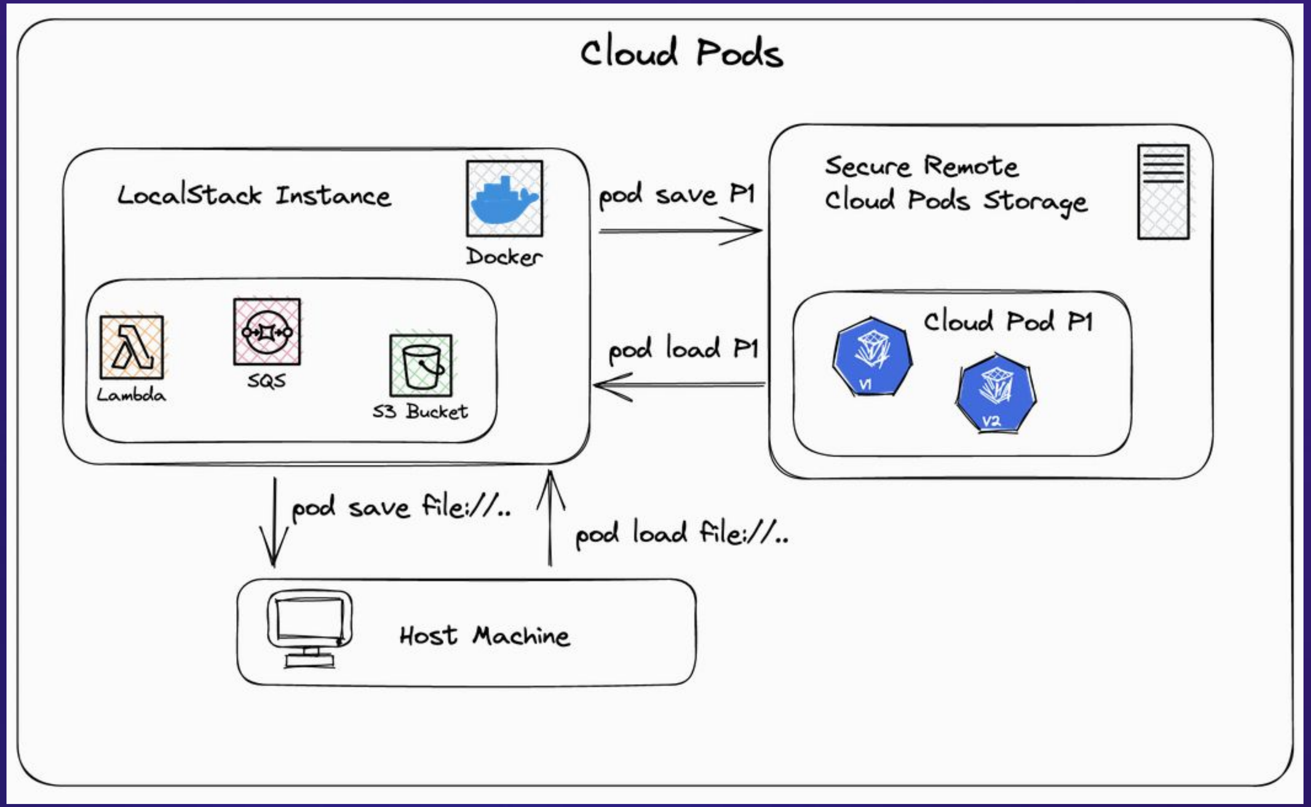
- Mini cloud operating system
  - Process management
  - File system abstractions
  - Event processing
  - Schedulers
  - Inter-service communication
  - Log management
  - ...
- Main design goals: lightweight, easy-to-use, cross-platform compatible

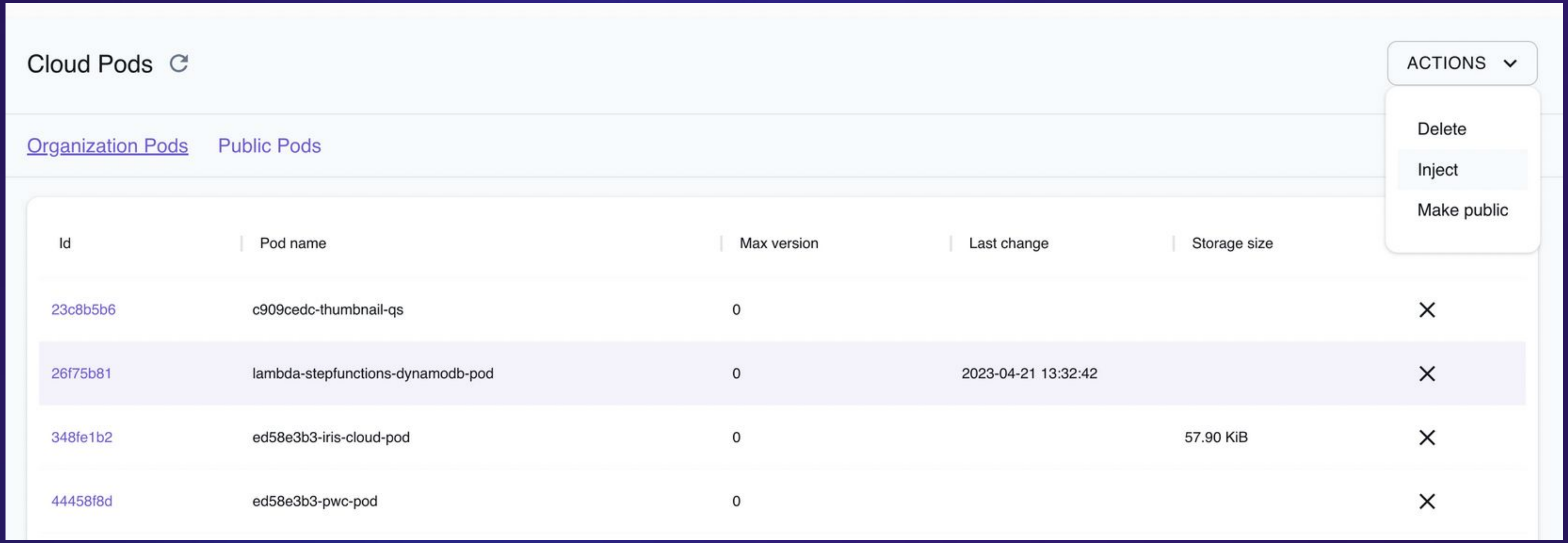


LocalStack Enterprise - Cloud Pods Storage

https://docs.localstack.cloud/user-guide/tools/cloud-pods

- Convenient use of **Cloud Pods**
  - Taking a snapshot of the state of a LocalStack instance
  - Storing and versioning in managed storage
  - Facilitates **team collaboration**
- Secure Managed Storage
  - Secure encrypted storage buckets per customer
  - Up to 5GB storage per user included
- Alternative storage backends available for on-prem
  - E.g., OCI registries, FTP servers, git repos
- Easy sharing within the team
  - Enable organization-wide access to pods
  - Browse and inject pods directly from the Web app

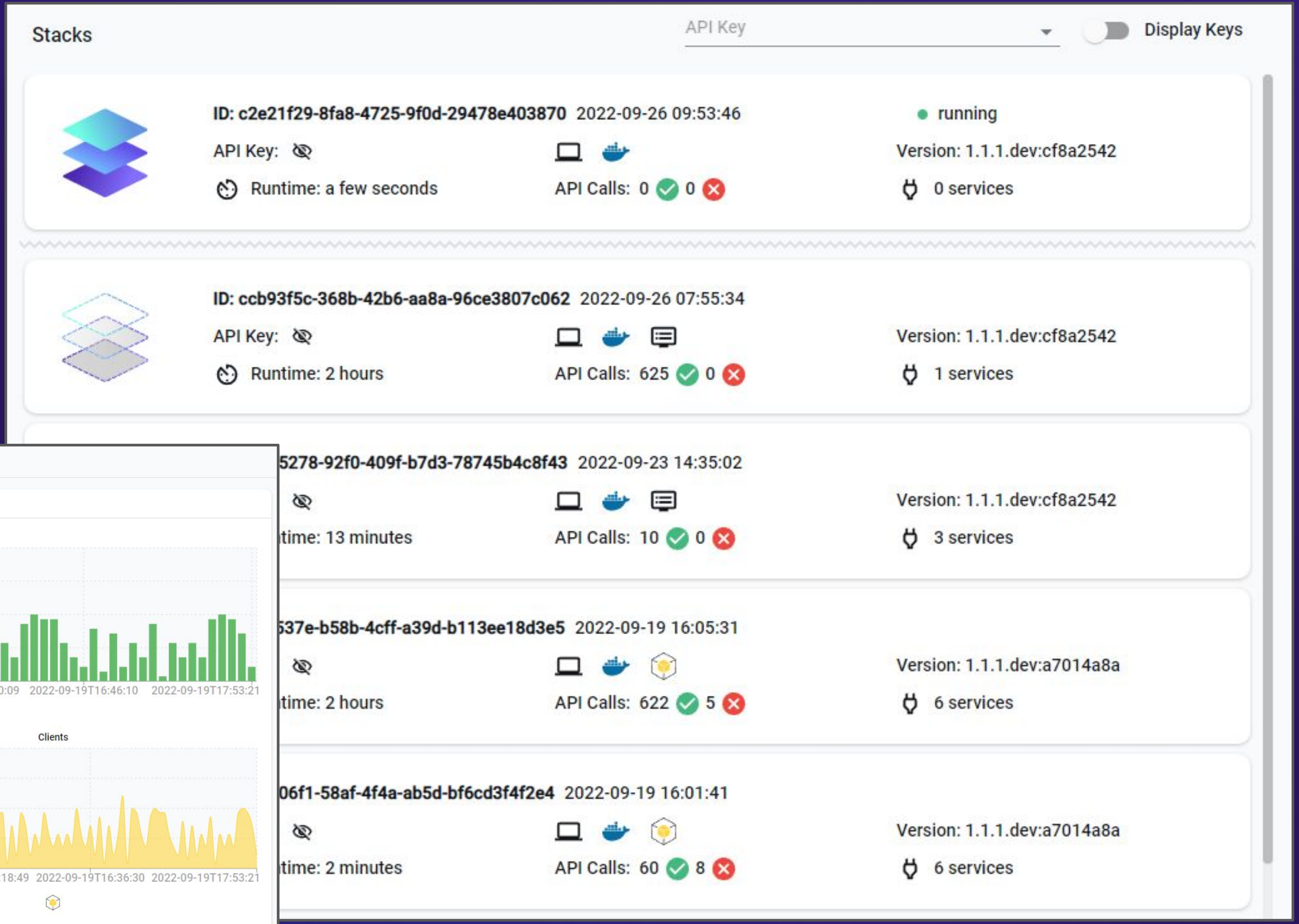




Id	Pod name	Max version	Last change	Storage size	
23c8b5b6	c909cedc-thumbnail-qs	0			✕
26f75b81	lambda-stepfunctions-dynamodb-pod	0	2023-04-21 13:32:42		✕
348fe1b2	ed58e3b3-iris-cloud-pod	0		57.90 KiB	✕
44458f8d	ed58e3b3-pwc-pod	0			✕

LocalStack Enterprise - Stack Insights

- Inspect the current and historical state of LocalStack instances
- Fine-grained access insights and debug logs



Stacks

API Key ▾ Display Keys

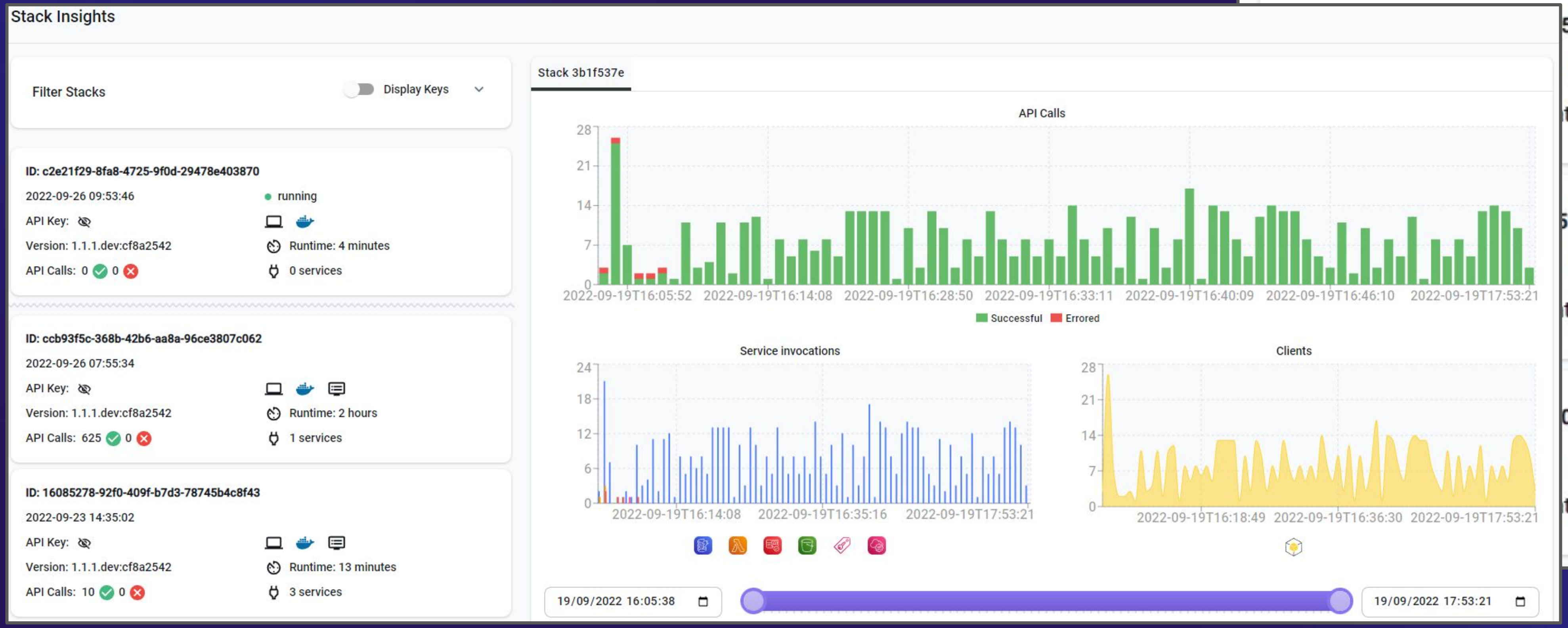
ID: c2e21f29-8fa8-4725-9f0d-29478e403870 2022-09-26 09:53:46 ● running
API Key: Version: 1.1.1.dev:cf8a2542
Runtime: a few seconds API Calls: 0 ✓ 0 ✗ 0 services

ID: ccb93f5c-368b-42b6-aa8a-96ce3807c062 2022-09-26 07:55:34
API Key: Version: 1.1.1.dev:cf8a2542
Runtime: 2 hours API Calls: 625 ✓ 0 ✗ 1 services

5278-92f0-409f-b7d3-78745b4c8f43 2022-09-23 14:35:02
Version: 1.1.1.dev:cf8a2542
time: 13 minutes API Calls: 10 ✓ 0 ✗ 3 services

537e-b58b-4cff-a39d-b113ee18d3e5 2022-09-19 16:05:31
Version: 1.1.1.dev:a7014a8a
time: 2 hours API Calls: 622 ✓ 5 ✗ 6 services

06f1-58af-4f4a-ab5d-bf6cd3f4f2e4 2022-09-19 16:01:41
Version: 1.1.1.dev:a7014a8a
time: 2 minutes API Calls: 60 ✓ 8 ✗ 6 services



Stack Insights

Filter Stacks Display Keys

ID: c2e21f29-8fa8-4725-9f0d-29478e403870
2022-09-26 09:53:46 ● running
API Key:
Version: 1.1.1.dev:cf8a2542 Runtime: 4 minutes
API Calls: 0 ✓ 0 ✗ 0 services

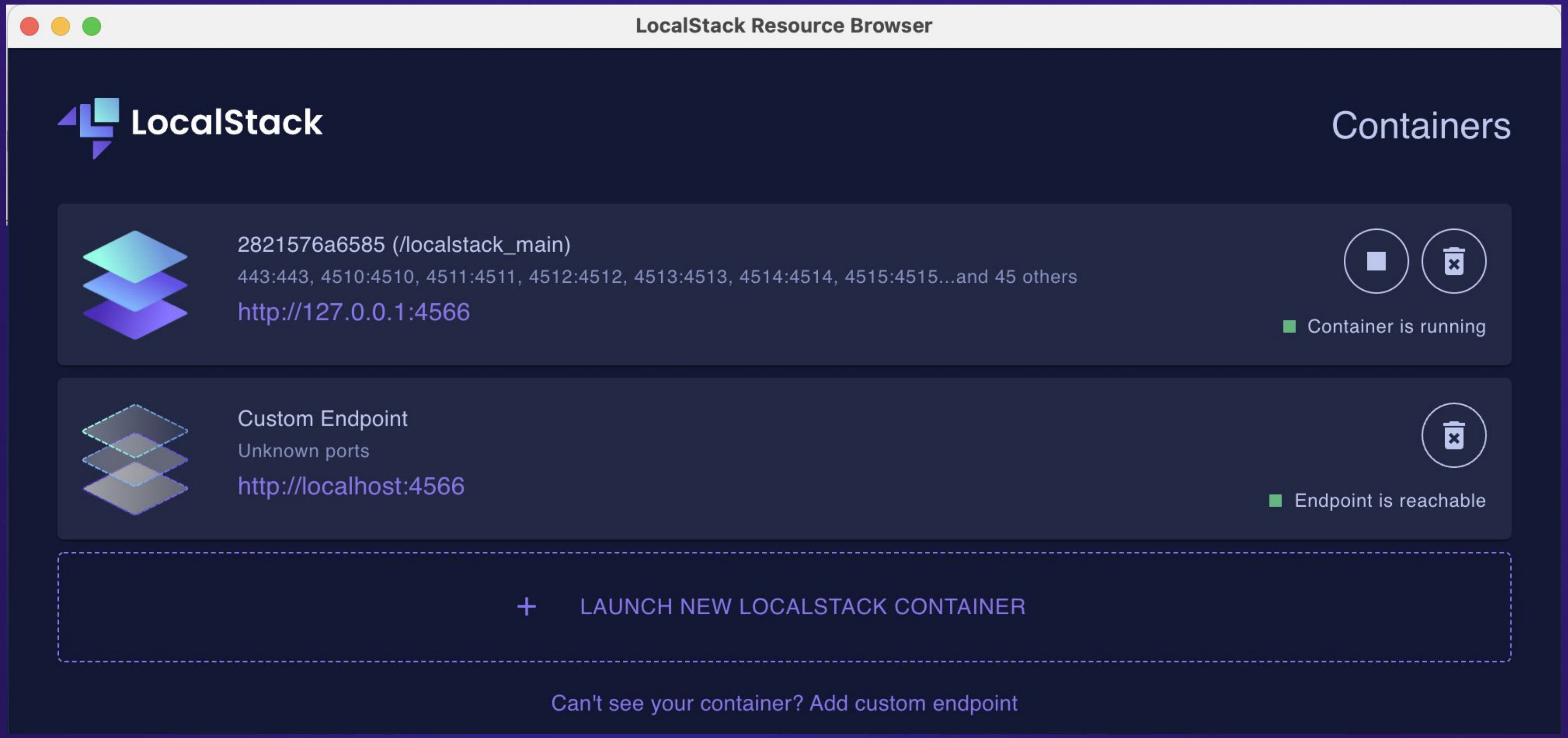
ID: ccb93f5c-368b-42b6-aa8a-96ce3807c062
2022-09-26 07:55:34
API Key:
Version: 1.1.1.dev:cf8a2542 Runtime: 2 hours
API Calls: 625 ✓ 0 ✗ 1 services

ID: 16085278-92f0-409f-b7d3-78745b4c8f43
2022-09-23 14:35:02
API Key:
Version: 1.1.1.dev:cf8a2542 Runtime: 13 minutes
API Calls: 10 ✓ 0 ✗ 3 services

localstack localstack localstack-cloud localstack.cloud

LocalStack Enterprise - Desktop App

- Manage LocalStack instances
  - Starting/stopping/monitoring Docker containers
  - Control local or remote instances
- Resource Browser
  - Simplified version of the AWS Console, running locally
  - Browse resources locally
  - Manage S3 buckets, Lambda functions, DynamoDB tables, etc
- Convenient Desktop app for local/offline use
  - Alternative to our hosted Web application


LocalStack Resource Browser

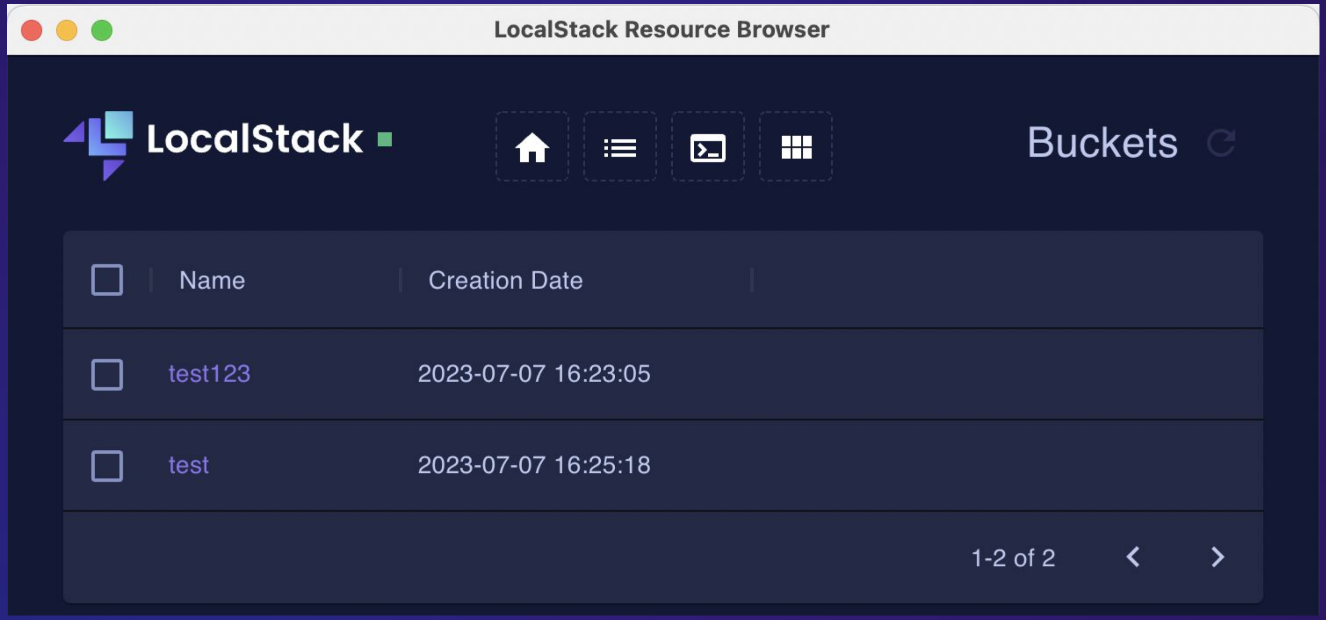
LocalStack Containers

2821576a6585 (/localstack_main)
443:443, 4510:4510, 4511:4511, 4512:4512, 4513:4513, 4514:4514, 4515:4515...and 45 others
http://127.0.0.1:4566
Container is running

Custom Endpoint
Unknown ports
http://localhost:4566
Endpoint is reachable

+ LAUNCH NEW LOCALSTACK CONTAINER

Can't see your container? Add custom endpoint


LocalStack Resource Browser

LocalStack Buckets

Name	Creation Date
test123	2023-07-07 16:23:05
test	2023-07-07 16:25:18

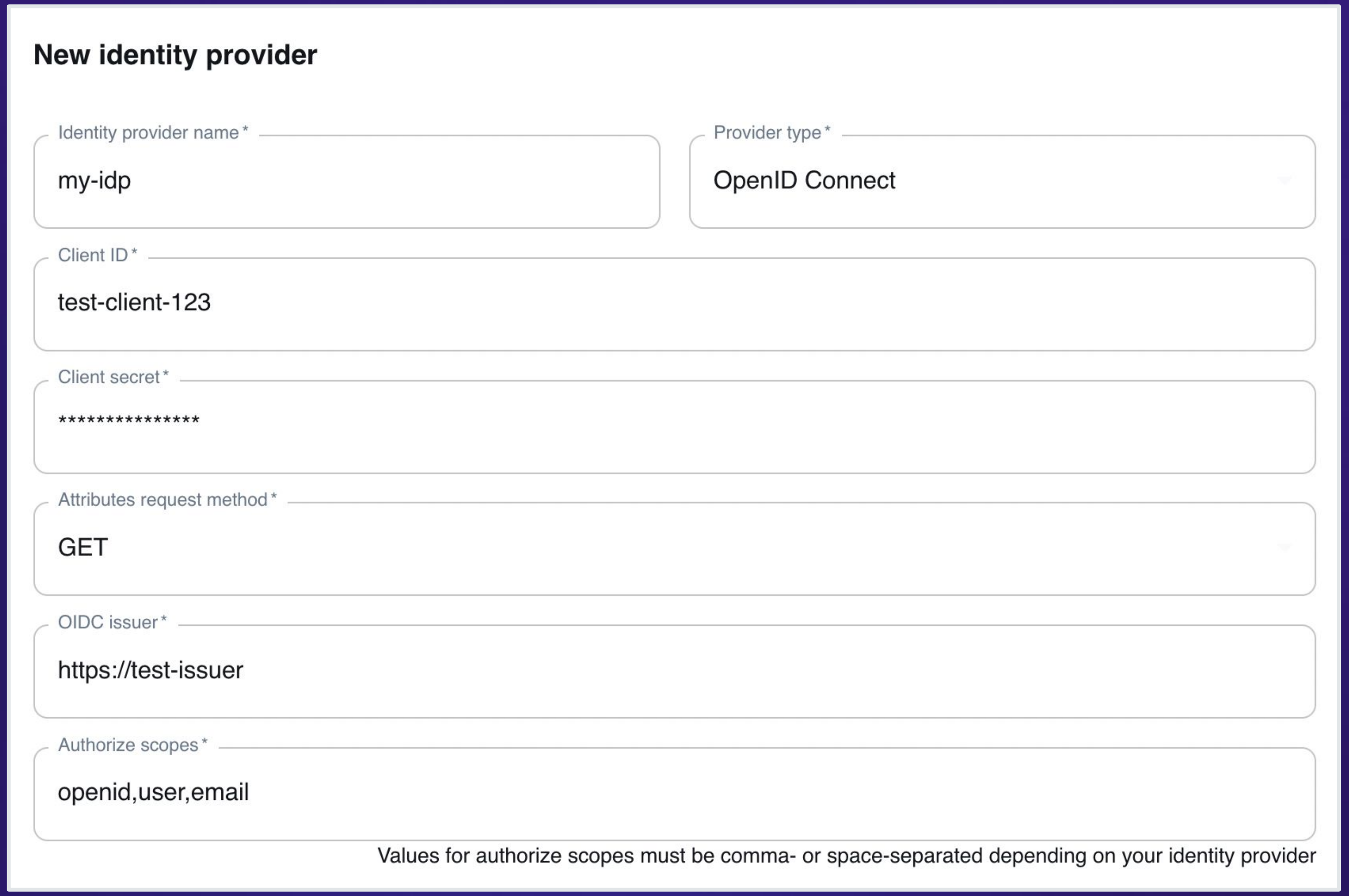
1-2 of 2

LocalStack Enterprise - SSO Support

- Single Sign-On support based on standard protocols
  - Open ID Connect (OIDC)
  - Security Assertion Markup Language (SAML)

- Easy integration with Azure Active Directory (AD), Okta, AWS Cognito, and other SSO providers
  - See docs for more details: https://docs.localstack.cloud/user-guide/web-application/single-sign-on

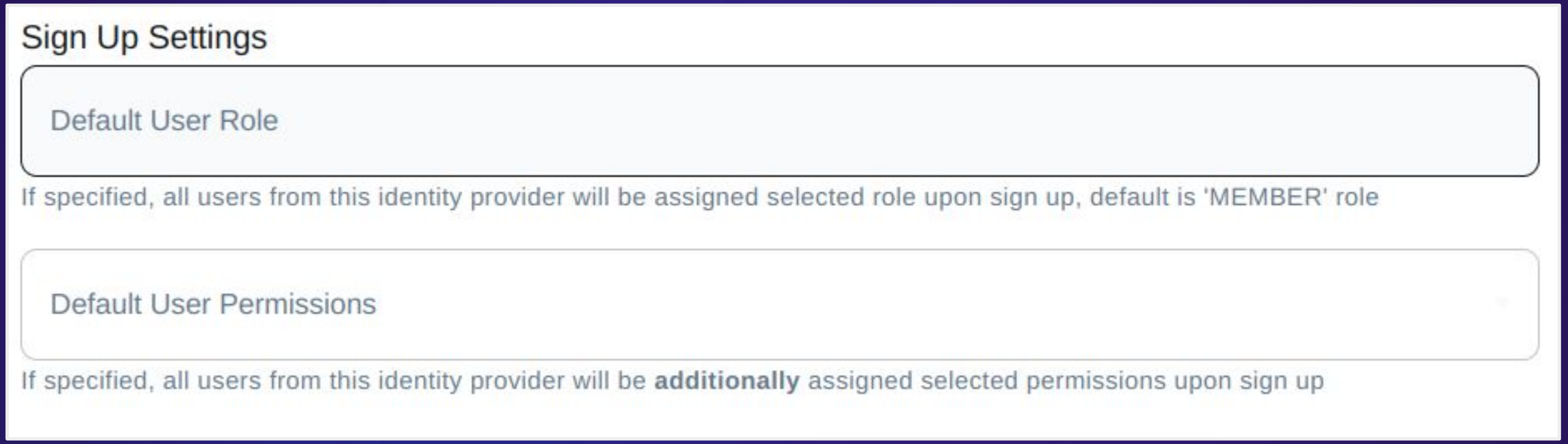
- Configurable default user roles and permissions
  - Predefined roles like *member* or *admin*
  - Fine-grained permissions for API key management, cloud pods usage, etc



New identity provider

Identity provider name *	my-idp
Provider type *	OpenID Connect
Client ID *	test-client-123
Client secret *	***************
Attributes request method *	GET
OIDC issuer *	https://test-issuer
Authorize scopes *	openid,user,email

Values for authorize scopes must be comma- or space-separated depending on your identity provider



Sign Up Settings

Default User Role

If specified, all users from this identity provider will be assigned selected role upon sign up, default is 'MEMBER' role

Default User Permissions

If specified, all users from this identity provider will be **additionally** assigned selected permissions upon sign up

LocalStack Enterprise - Extensions

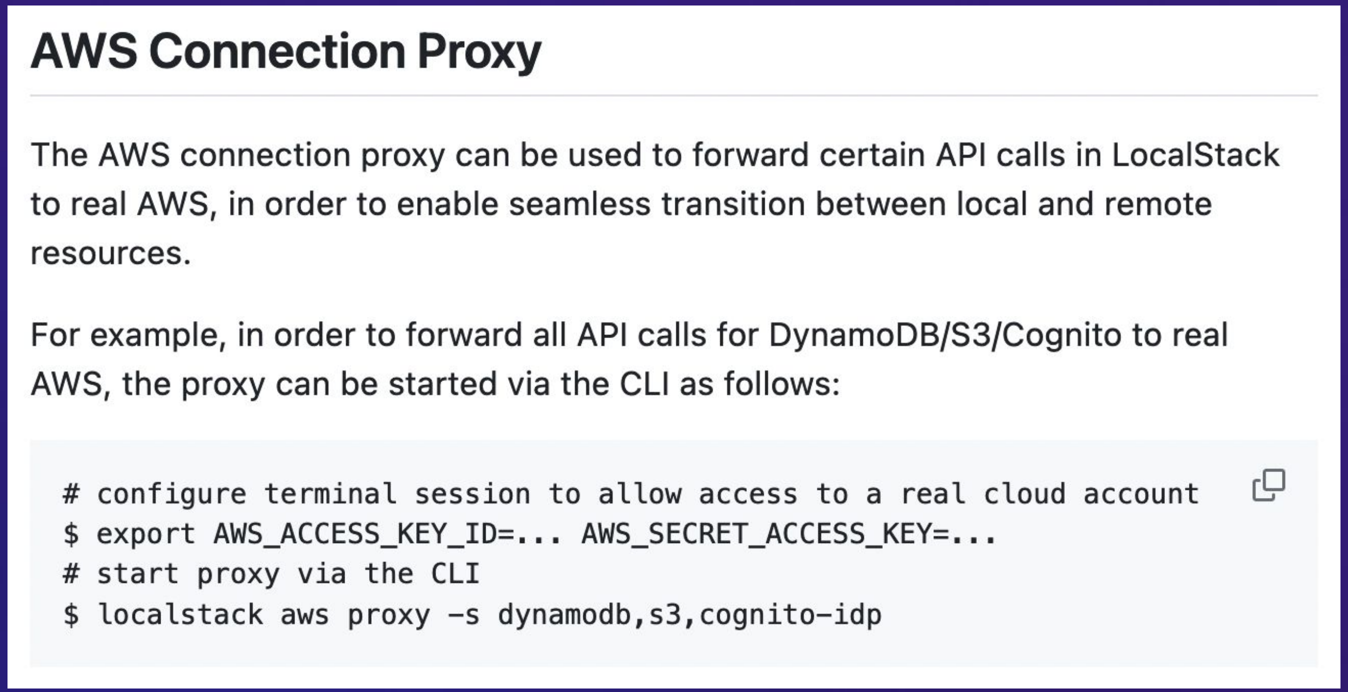
- Example: **AWS Replicator (Proxy) Extension**
  - Bridging the gap between local execution and cloud resources

- Enables “hybrid scenarios”
  - → part of the application stack running locally, partly accessing remote resources in the cloud
  - Simplifies development and testing in large-scale scenarios

- Integrates seamlessly with all LocalStack Tooling
  - E.g., `awslocal`, `tflocal` (Terraform), `cdklocal` (CDK), etc

- Fine-grained configuration
  - Which resources are accessed locally or remotely
  - Either on the service- or on the resource-level

- Other custom extensions available on-demand
  - E.g., fault injection testing, third-party emulators, etc

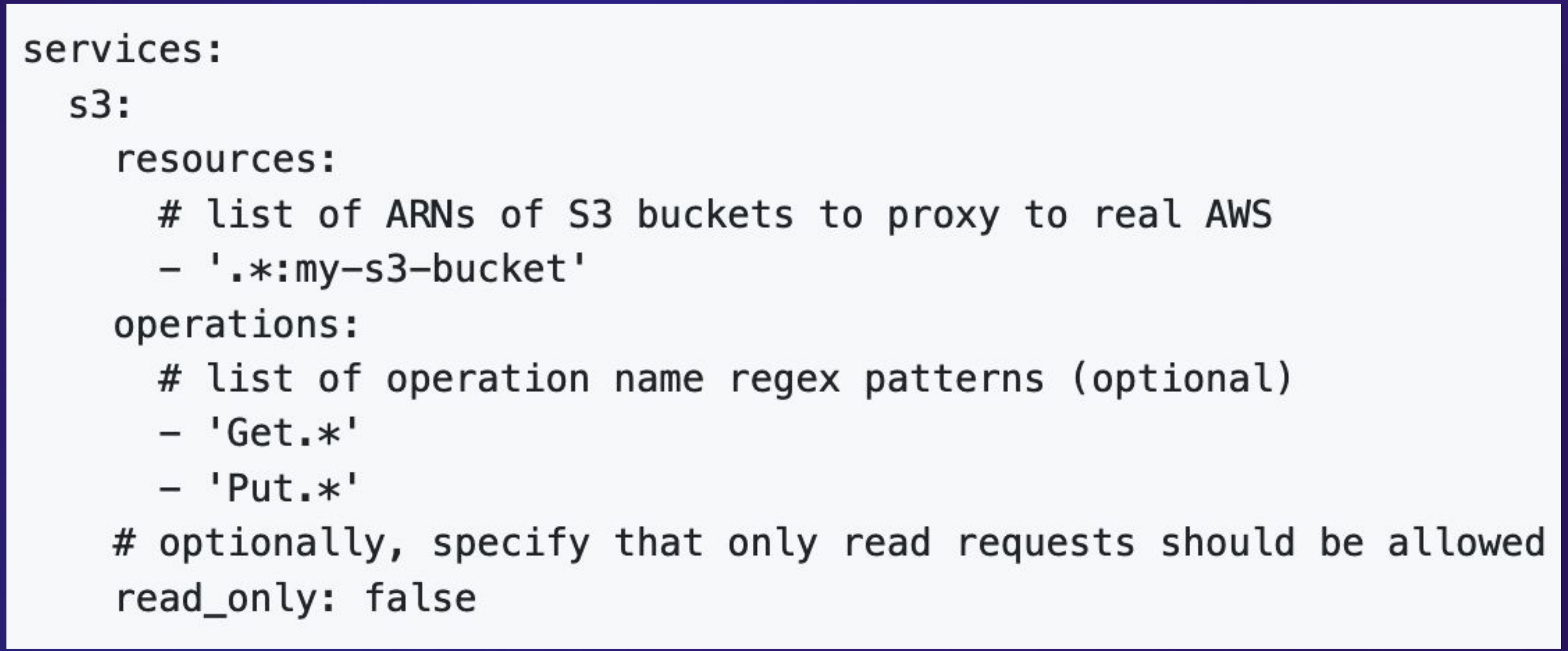


AWS Connection Proxy

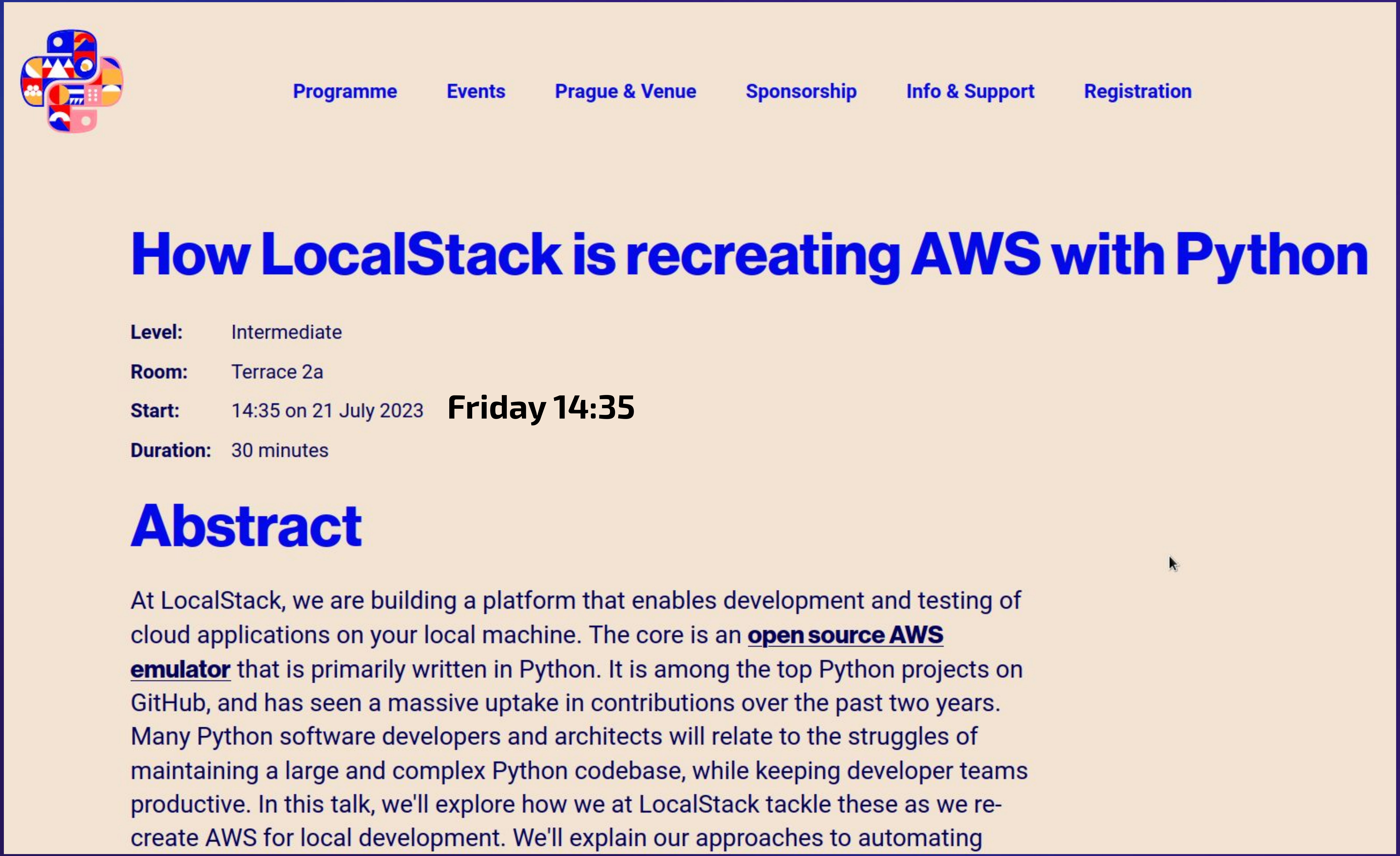
The AWS connection proxy can be used to forward certain API calls in LocalStack to real AWS, in order to enable seamless transition between local and remote resources.

For example, in order to forward all API calls for DynamoDB/S3/Cognito to real AWS, the proxy can be started via the CLI as follows:

```
# configure terminal session to allow access to a real cloud account
$ export AWS_ACCESS_KEY_ID=... AWS_SECRET_ACCESS_KEY=...
# start proxy via the CLI
$ localstack aws proxy -s dynamodb,s3,cognito-idp
```



```
services:
  s3:
    resources:
      # list of ARNs of S3 buckets to proxy to real AWS
      - '.*:my-s3-bucket'
    operations:
      # list of operation name regex patterns (optional)
      - 'Get.*'
      - 'Put.*'
    # optionally, specify that only read requests should be allowed
    read_only: false
```


Programme Events Prague & Venue Sponsorship Info & Support Registration

How LocalStack is recreating AWS with Python

Level:	Intermediate
Room:	Terrace 2a
Start:	14:35 on 21 July 2023
Duration:	30 minutes

Friday 14:35

Abstract

At LocalStack, we are building a platform that enables development and testing of cloud applications on your local machine. The core is an **open source AWS emulator** that is primarily written in Python. It is among the top Python projects on GitHub, and has seen a massive uptake in contributions over the past two years. Many Python software developers and architects will relate to the struggles of maintaining a large and complex Python codebase, while keeping developer teams productive. In this talk, we'll explore how we at LocalStack tackle these as we re-create AWS for local development. We'll explain our approaches to automating