Best Practices for Testing and Debugging of Cloud Applications

Dr. Waldemar Hummer
LocalStack GmbH
waldemar@localstack.cloud

Outline

- Introduction
- Background: Cloud vs Traditional Software Development
- Challenges in Today’s Cloud Development
- Repeatability, IaC, Ephemeral Environments
- Cloud Testing Approaches
  - Mocking, Emulation, Remote, Hybrid Execution
- Advanced Use Cases for Cloud Emulation
- Conclusion

Cloud vs Traditional Software Development

Cloud App Development

- Cloud environments provide a number of **managed services**
  - Well-defined interfaces (input/output messages) and semantics
  - Using APIs with some well-defined protocol (e.g., JSON/REST over HTTP)

- Services for **different purposes** / concerns
  - Compute, e.g.:
    - Function-as-a-Service (e.g., AWS Lambda)
    - Containerized Applications (e.g., Docker containers, Kubernetes pods)
    - Virtual Machines (e.g., AWS Elastic Compute Cloud (EC2))
  - Databases, e.g.:
    - Relational DBs, Graph DBs, Key-Value stores, etc.
  - Messaging, e.g.:
    - Queueing services, Pub/Sub systems, Streaming systems (e.g., Kafka)
  - Ingest, e.g.:
    - API Gateways, GraphQL APIs, Content Distribution Networks (CDNs), etc
  - ...

Raising the Abstraction Level

- Simplification - users can focus on developing application logic
  - “Serverless” computing - removing the necessity to manage/install servers
- Example: Lambda functions on AWS
  - Simple Lambda handler that prints the invocation event and returns it to the client

```
1  exports.handler = async (event, context) => {
2      console.log("Hello from Lambda function!");
3      console.log("Received invocation event:", event);
4      return {echo: event};
5  };
```

- Deployment:

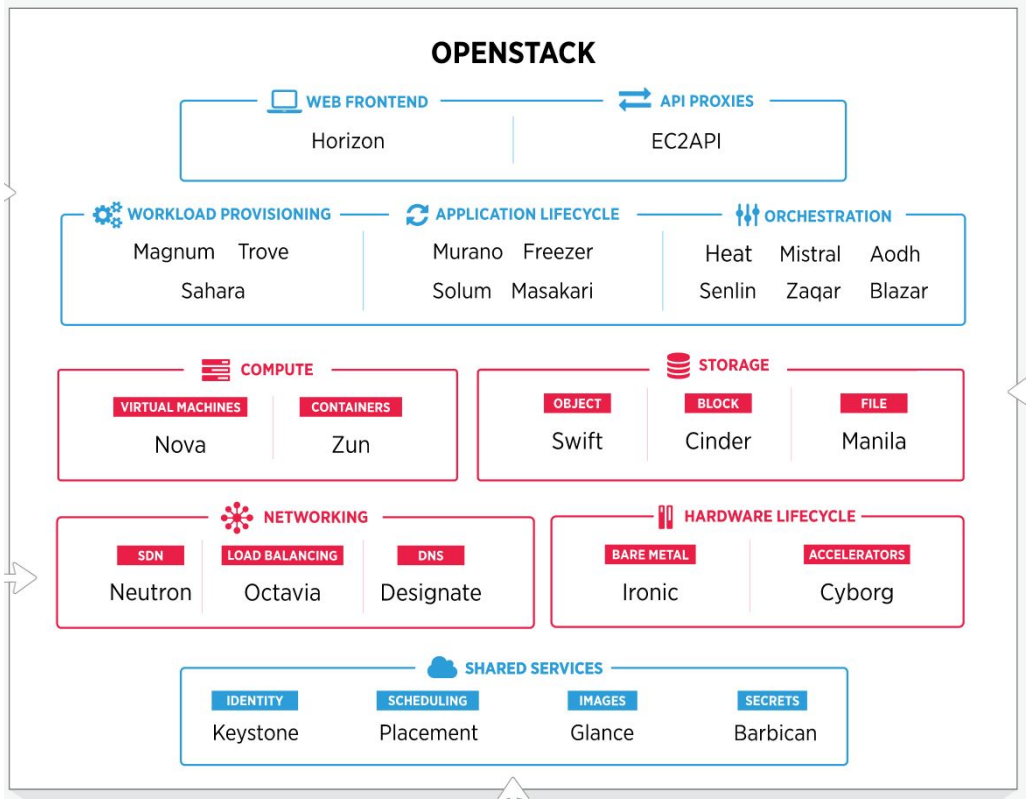
```
$ zip lambda.zip lambda.js
$ aws lambda create-function --function-name func1 --runtime nodejs14.x \
  --role arn:aws:iam:... --handler lambda.handler --zip-file fileb://lambda.zip
$ aws lambda invoke --function-name func1 /tmp/tmp.out
```

Cloud Operating System

- Internally, the Cloud is a higher-level OS

Traditional OS	Cloud OS
Processes	FaaS, Containers, VMs
Disk controllers	Storage services
Network I/O	VPCs, API/NAT Gateways
Scheduler	Event Bus
Access Control	IAM users, roles, policies
IPC / signals	API calls / notifications

Example: OpenStack



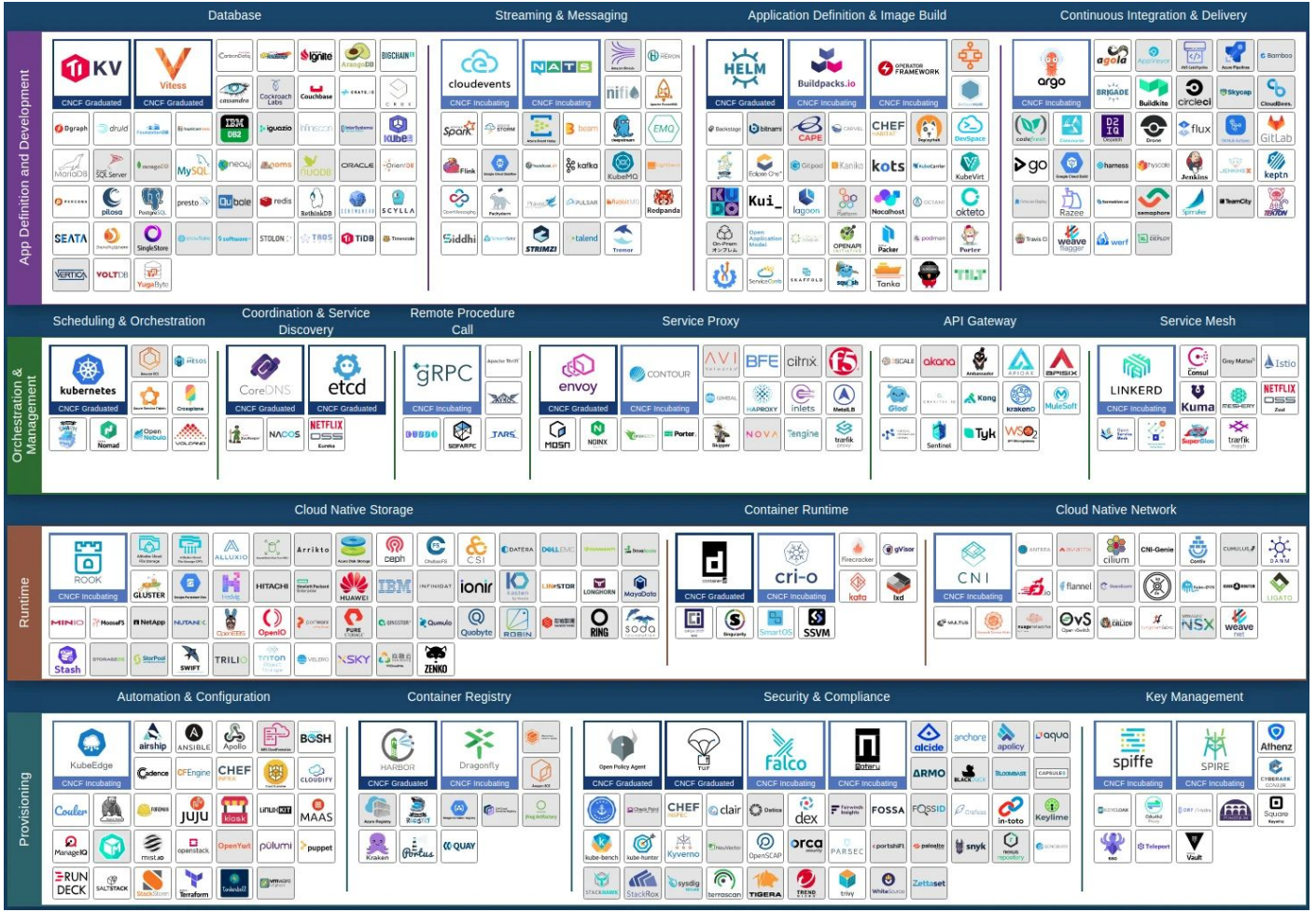
https://www.openstack.org/software

Cloud Native Computing Foundation (CNCF)

→ Large landscape of services

→ Split up into main areas (App dev., orchestration, runtime, provisioning, ...)

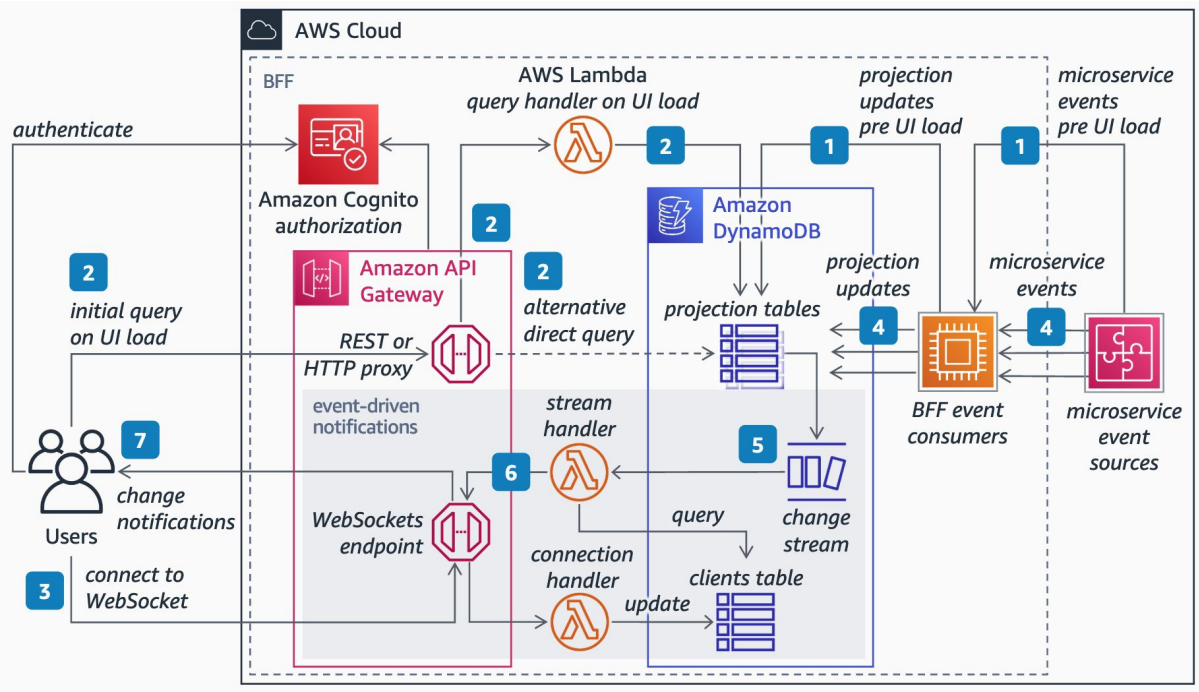
→ strong focus on Kubernetes-based and cloud agnostic cooling



https://landscape.cncf.io

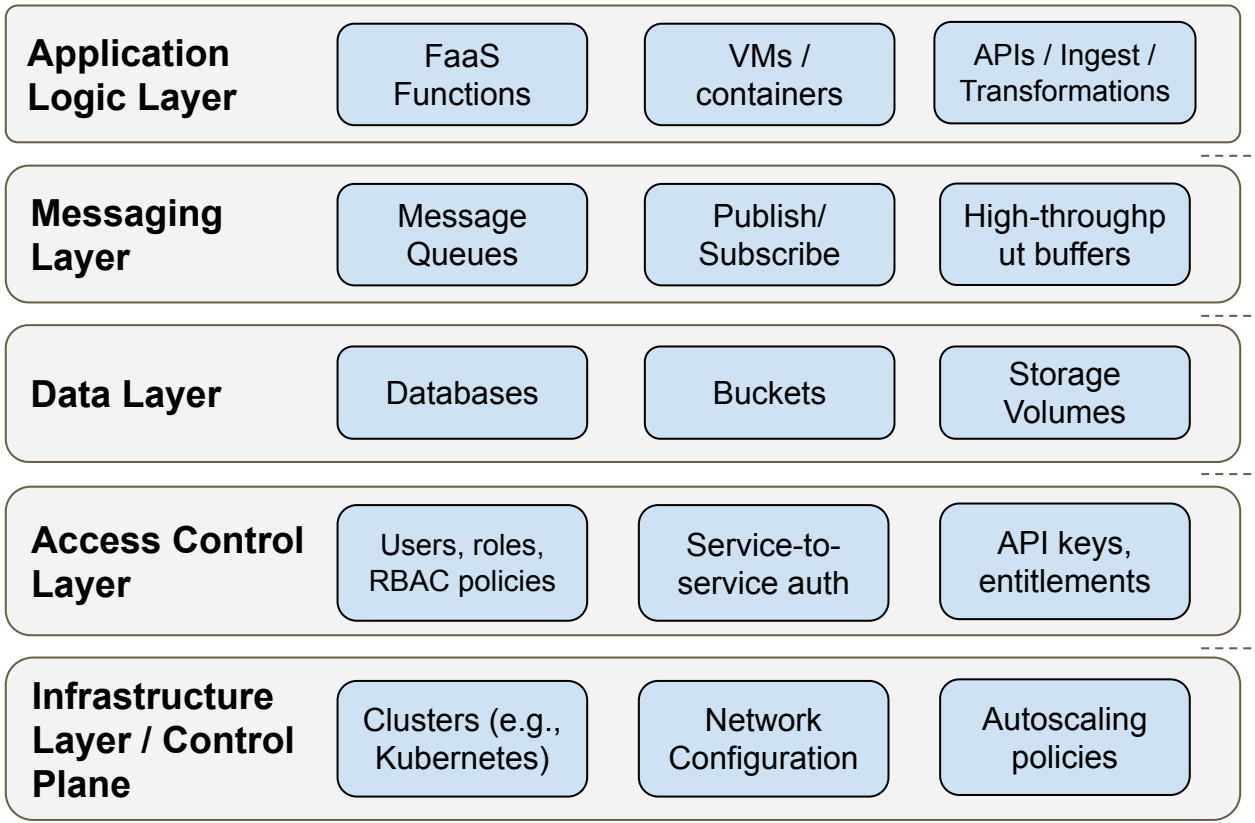
Sample Cloud Application

- Reference Architecture: **Backend for frontend (BFF) using API Gateway**



https://aws.amazon.com/architecture/reference-architecture-diagrams

Cloud Application Layers (simplified)



Testing Focus

→ End-to-end app logic
→ Main user flows / interactions
→ Unit + integration tests

→ Different delivery semantics
→ Termination conditions (e.g., avoiding infinite update loops)

→ Basic CRUD operations
→ Data integrity
→ Data streams updates

→ Assert access policies
→ Positive and negative tests
→ (Note: can become a hindrance to test other layers)

→ Correct provisioning logic
→ Handle out-of-band changes
→ Resource Teardown/cleanup

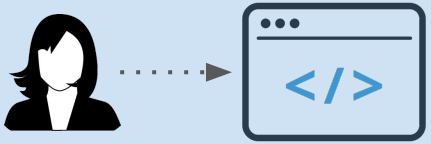
Approaches for Testing & Debugging

	Traditional App. Development	Cloud Computing
Dev. feedback cycles	Local code compilation	Uploading code to the cloud
Debugging	Setting local breakpoints	Based on log outputs, tracing information
Local Testing	Mocking of dependencies	Often testing in the real environment
Distributed Execution	Monolithic / larger components	Inherently distributed / event-based logic
State Inspection	Memory dumps, profiling, …	Logs, tracing, monitoring metrics
Reproducibility	Restart app & restore/inject state	Logs & API calls (harder to restore state)
Security & Auth.	Often tackled by a middleware	Inherent part of service-to-service comm.

Challenges in Today's Cloud Development

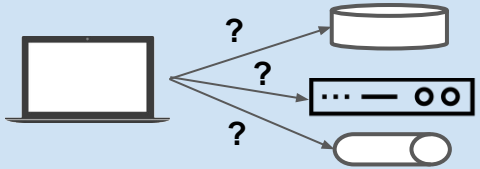
Context: One day in the life of a Cloud developer

①



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

②

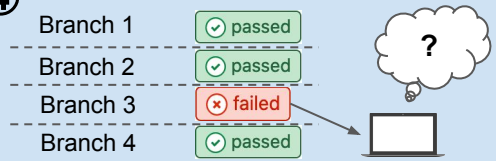


Developing on her local machine, she realizes that there are lots of **dependencies** with resources in the **cloud** (DBs, VMs, MQs, S3, …)

③

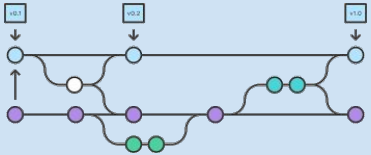
Alice realizes that the dev&test loop is extremely **slow and tedious**. Every local change needs to be packaged and **uploaded** to the cloud for testing.

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).

[1] https://www.flexera.com/blog/application-readiness/cloud-computing-trends-2021-state-of-the-cloud-report/

Digression (a bit controversial*): Cloud being compared to the mainframe era

Hackernews user “jiggawatts”

“The cloud is the new time-share mainframe. Programming in the 1960s to 80s was like this too. You'd develop some program in isolation, unable to properly run it. You "submit" it to the system, and it would be scheduled to run along with other workloads. You'd get a printout of the results back hours later, or even tomorrow. Rinse and repeat.

This work loop is incredibly inefficient, and was replaced by development that happened entirely locally on a workstation. This dramatically tightened the edit-compile-debug loop, down to seconds or at most minutes. Productivity skyrocketed, and most enterprises shifted the majority of their workload away from mainframes.

Now, in the 2020s, mainframes are back! They're just called "the cloud" now, but not much of their essential nature has changed other than the vendor name.”

* This discussion is a bit controversial - please take it with a grain of salt :)

https://hacker-news.news/post/26855037

Digression (a bit controversial): Cloud being compared to the mainframe era

Hackernews user “jiggawatts”*

“The cloud, just like mainframes:

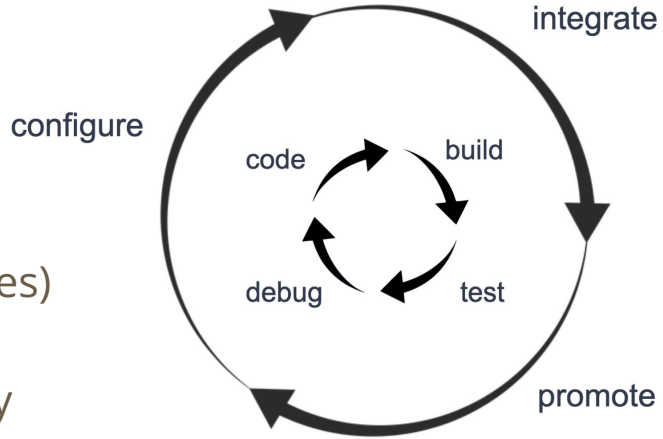
- Does not provide all-local workstations. The only full-fidelity platform is the shared server.
- Is closed source. Only Amazon provides AWS. Only Microsoft provides Azure. Only Google provides GCP. You can't peer into their source code, it is all proprietary and even secret.
- Has a poor debugging experience. Shared platforms can't generally allow "invasive" debugging for security reasons. Their sheer size and complexity will mean that your visibility will always be limited. You'll never been able to get a stack trace that crosses into the internal calls of the platform services like S3 or Lambda. Contrast this with typical debugging where you can even trace into the OS kernel if you so choose.
- Are generally based on the "print the logs out" feedback mechanism, with all the usual issues of mainframes such as hours-long delays.”

https://hacker-news.news/post/26855037

Inner vs Outer Dev Loop

- **Inner Dev Loop**
  - Quick iterations, frequent changes, mostly on the local machine
  - Tools: IDE, debugger, break points, memory dumps, execution traces
- **Outer Dev Loop**
  - Infrequent changes, testing the integration, long-running tests
  - Often executed via automated builds in a CI/CD system

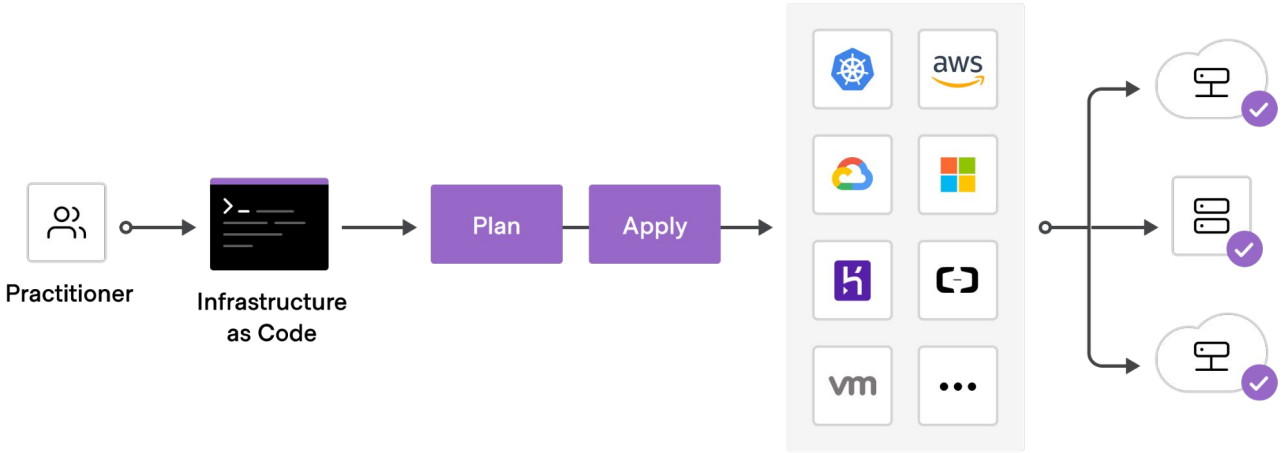
- Achieving an efficient inner dev loop is the "holy grail" for efficient SWE, but can be challenging, e.g.:
  - instant feedback for application changes
  - managing dependencies (e.g. utility microservices)
  - In practice, larger organizations often employ dedicated DevX teams to optimize dev efficiency



Repeatability, IaC, Ephemeral Environments

Infrastructure-as-Code

- IaC has become popular with the DevOps movement
  - Applying software engineering best practices to infrastructure management
  - Shielding off production systems from any manual changes
- Resources are defined in a declarative way
  - E.g., Terraform: creates a **plan**, which is then **applied** to create the resources



IaC - Terraform

- Ability to define resources declaratively
  - E.g., SQS queues, Lambda functions, etc.
- Existing resources are automatically determined by TF
- Note: Also IaC scripts need thorough testing!
  - Often the application logic (e.g., Lambda function) is actually not that relevant → even more important to have quick feedback cycles

```
1  # SQS queue
2  resource "aws_sqs_queue" "test_queue" {
3    name                      = "tf-sqs-queue-1"
4  }
5
6  # Lambda function
7  resource "aws_lambda_function" "test_lambda" {
8    filename      = "lambda.zip"
9    function_name = "terraform_test_lambda"
10   role          = "${aws_iam_role.lambda.arn}"
11   handler       = "lambda.hello"
12   runtime       = "nodejs12.x"
13   source_code_hash = "${filebase64sha256("lambda.zip")}"
14 }
15
16 # Lambda event source mapping from SQS queue
17 resource "aws_lambda_event_source_mapping" "test_mapping1" {
18   event_source_arn  = aws_sqs_queue.test_queue.arn
19   function_name     = aws_lambda_function.test_lambda.arn
20   starting_position = "LATEST"
21 }
```

References to resource IDs/ARNs - used internally by TF to build the dependency graph

Cloud SDKs

- Software Development Kits (SDKs) used to interact with the Cloud
- Available for different programming languages
- Example: creating an S3 bucket in AWS
  - Python:

```python
s3 = boto3.client("s3", region_name="us-east-1")
s3.create_bucket(Bucket="mybucket1")
```

  - Java:

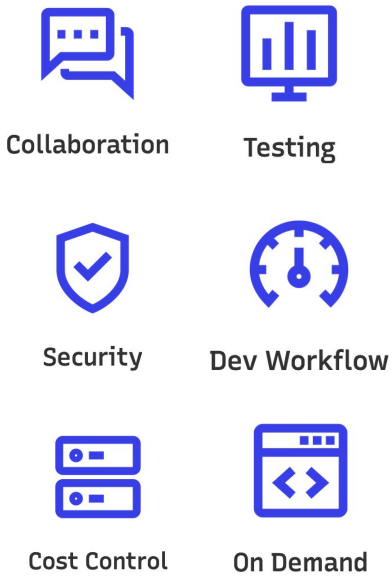
```java
AmazonS3 s3 = AmazonS3ClientBuilder.standard().withRegion("us-east-1").build();
Bucket bucket = s3.createBucket("mybucket1");
```

  - Golang:

```golang
sess, _ := session.NewSession(&aws.Config{Region: aws.String("us-east-1")})
client := s3.New(sess)
b, _ := client.CreateBucket(&s3.CreateBucketInput{Bucket: aws.String("mybucket1")})
```

Ephemeral Environments

- Short-lived environments that are created for a certain purpose
  - User acceptance testing, UI layout review, quick experimentation, ...
- Critical in a testing context
  - Running Continuous Integration (CI) builds on every code change
  - Infrastructure needs to be frequently created and teared down
  - Providing app previews - e.g., show an updated version of a Web UI to review changes
- Can be achieved with Infrastructure-as-Code scripts
  - Simplest case: applying infrastructure changes against a clean/fresh environment
  - Requires some logic/parametrization when applied against an existing environment that contains resources (e.g., to avoid naming conflicts)
- Becoming quite popular in the container/Kubernetes space
  BUT: still rarely available for managed cloud services!

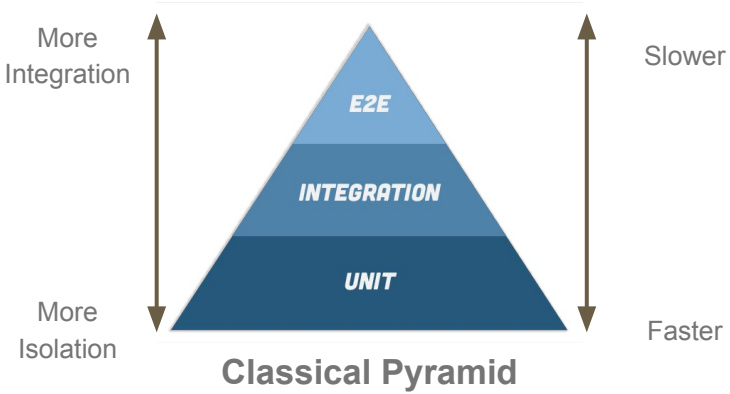


https://ephemeralenvironments.io

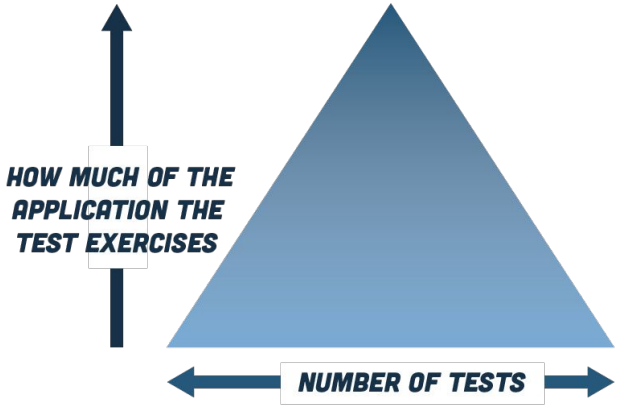
Cloud Testing Approaches

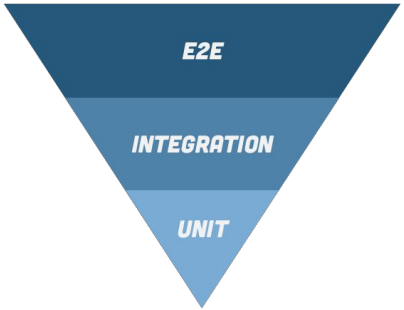
https://blog.ncrunch.net/post/testing-pyramid-automated-testing.aspx
https://martinfowler.com/articles/practical-test-pyramid.html

The Testing Pyramid

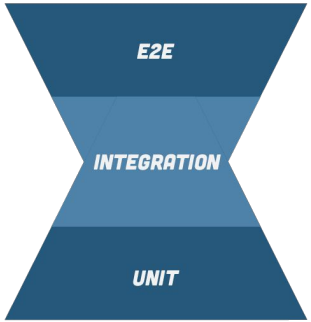


Classical Pyramid

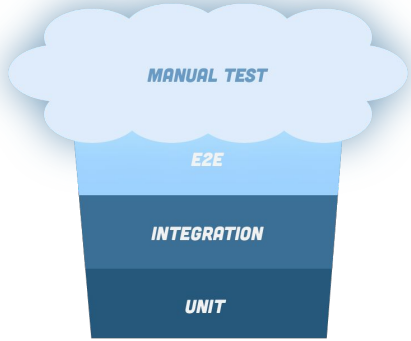




"Ice Cream Cone"



"Hourglass"



"Cupcake"

Cloud Testing Approaches

- Different strategies for developing application logic and executing tests

Local Testing

Hybrid Setups

Remote Testing

Mocking

→ Create mocks, fixtures that provide test data

Pros:
→ quick dev loops
→ fast test execution

Cons:
→ high effort
→ not reusable

Emulation

→ Emulate the behavior of cloud APIs locally

Pros:
→ quick dev loops
→ little adjustments in code

Cons:
→ hard to achieve full parity
→ resource constraints

Remote Cloud APIs

→ Run all tests against the real cloud environment

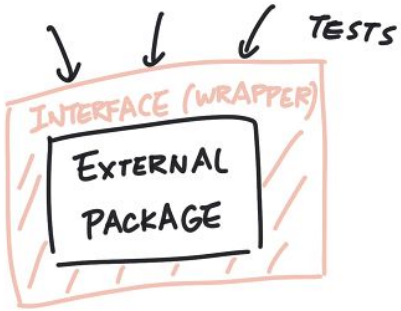
Pros:
→ full power of real cloud
→ include IAM/security early on

Cons:
→ slow dev loops
→ collaboration barriers
→ reduced debuggability

Mocking

- Frequently used method in software engineering and testing
  - Change the behavior of a certain piece of logic for the duration of a test
  - Lots of different frameworks for different programming languages
    - Python: pytest, Java: mockito,
- Cloud mocking: usually tackled on the SDK level

```python
import pytest
import boto3

@pytest.fixture
def s3_client(mocker):
    def mock_api_call(self, operation_name, kwarg):
        if operation_name == "ListBuckets":
            return {"Buckets": [{'Name': "test123"}]}
        ...
    mocker.patch("botocore.client.BaseClient._make_api_call", new=mock_api_call)
    return boto3.client("s3")

def test_s3(s3_client):
    result = s3_client.list_buckets()["Buckets"]
    assert result == [{'Name': "test123"}]
```



Mocking (2)

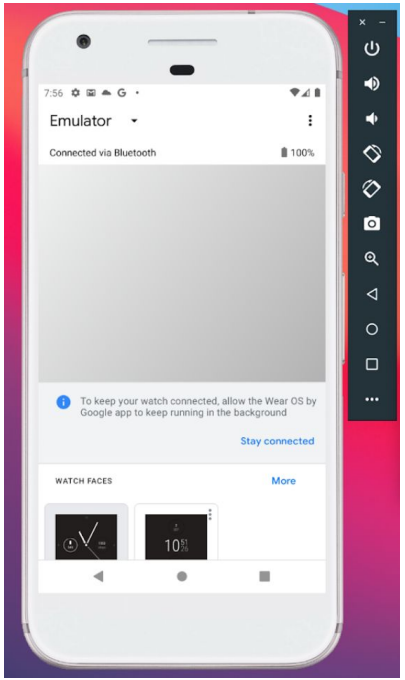
- Some Cloud SDKs even provide built-in mocking support
- e.g., botocore Stubber in the AWS Python SDK

```python
import boto3
from botocore.stub import Stubber

client = boto3.client('s3')
stubber = Stubber(client)
list_buckets_response = {
    "Owner": {
        "DisplayName": "name",
        "ID": "EXAMPLE123"
    },
    "Buckets": [{
        "CreationDate": "2016-05-25T16:55:48.000Z",
        "Name": "foo"
    }]
}
expected_params = {}
stubber.add_response('list_buckets', list_buckets_response, expected_params)

with stubber:
    response = client.list_buckets()

assert response == list_buckets_response
```

Based on: https://stackoverflow.com/questions/37143597/mocking-boto3-s3-client-method-python

Emulation

- Provide a representative version of the real system
  - Lower the barrier for development
  - Allows developing apps without actually owning the device
- Emulation has been popular in certain areas:
  - Mobile phone emulation (e.g., Android Studio)
  - Browser emulation (e.g., test Web apps for different browsers)
  - Embedded systems - abstractions for hardware components
  - Simulation - allows for creating different test scenarios
    - e.g., changing the system time, simulating faults, ...
- Increasingly also popular for cloud / managed services
  - Enables experimentation, easier integration with tests in CI pipelines
  - Can dramatically simplify and speed up testing (at least for certain scenarios)



Cloud Mocking and Emulation Libraries

- moto
  - Community project on Github that focuses on mocking the AWS SDK in Python
- LocalStack
  - Arguably the most advanced emulation platform currently out there (current focus on AWS)
  - Provides a platform (mini Cloud OS) to run users' cloud workloads on the local machine
  - Plugin system allows to easily plug in new service providers
- Tools by Cloud Providers
  - Cloud providers have published a few individual tools, but relatively fragmented:
  - AWS: StepFunctions Local, DynamoDB Local
  - GCP: emulators for bigtable, datastore, firestore, pubsub, spanner (mostly focused on DBs)
  - Azure: Storage Emulator (Blob, Queue, and Table services)
- Various smaller projects on Github that focus on individual cloud services, or generic mocking libraries

Simulation

- Cloud environments are large, dynamic, distributed systems
  - Lots of complex interactions constantly happening in parallel
  - Exposed to external user requests which can spike and fall (resulting in auto-scaling)
  - Subject to resource quotas, and other pricing optimizations (FinOps)

- Various faults can (and do!) happen at runtime
  - E.g., `ProvisionedThroughputExceeded` for throughput-constrained databases on AWS
  - Network partitions, IAM security policy enforcement issues, etc
  - Duplicate messages generated by services using *at-least-once* messaging semantics

- Chaos Engineering
  - Deliberately inject faults, to make the application logic more resilient
  - E.g. Chaos Mesh - a chaos engineering platform for Kubernetes (https://chaos-mesh.org)
    - fault injection for network, disk, file system, operating system, etc

Advanced Use Cases for Cloud Emulation

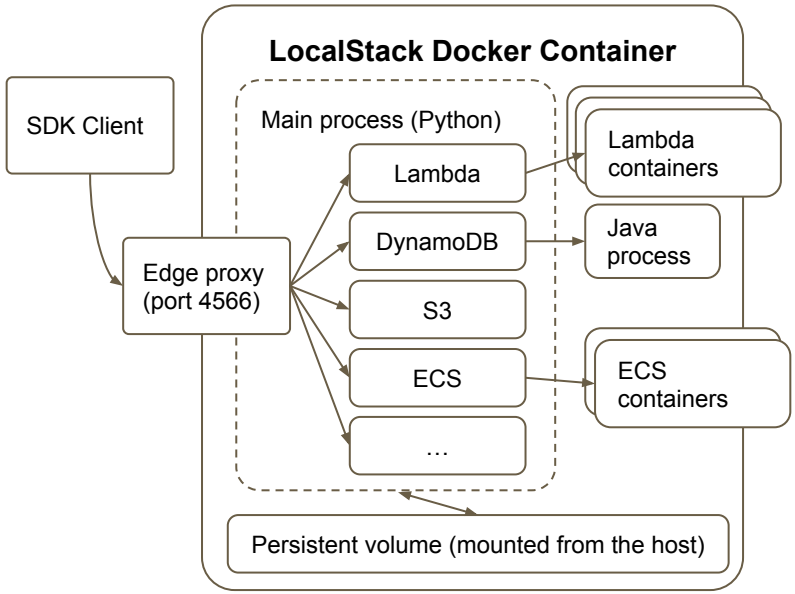
LocalStack

- Platform to emulate cloud environments on the local machine
  - Current focus on AWS, but gradually developing into a multi-cloud platform
- Very strong community traction (40k+ stars on Github)

- Shipped as a Docker image
  - Light-weight, easy to install

- Exposes APIs on a central port
  - AWS SDK clients can be configured to connect to http://localhost:4566

- State is kept in memory by default
  - Can also be persisted to disk

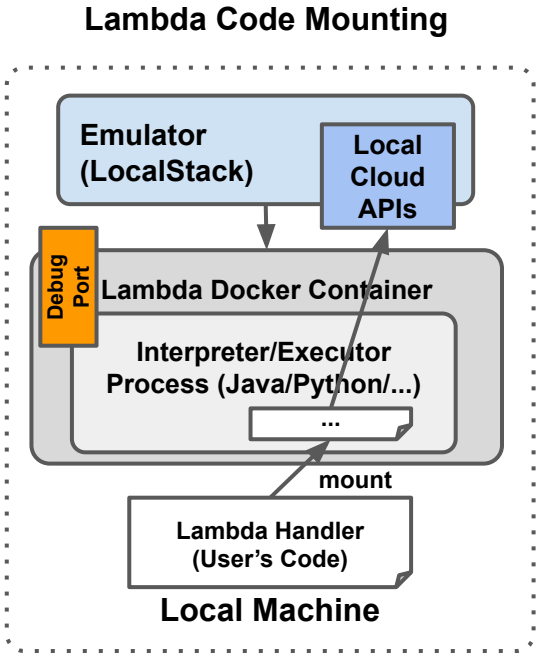


LocalStack Startup

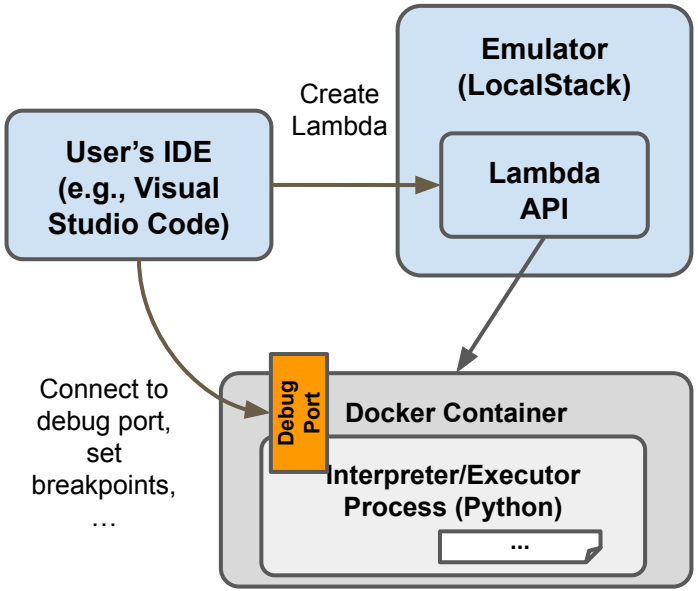
```
whummer@whu-mbp localstack-ext % localstack start
. .venv/bin/activate; PYTHONPATH=. localstack start --host


     __                     _______ __             __
    / /   ____  _________ _/ / ___// /_____ ______/ /__
   / /   / __ \/ ___/ __ `/ /\__ \/ __/ __ `/ ___/ //_/
  / /___/ /_/ / /__/ /_/ / /___/ / /_/ /_/ / /__/ ,<
 /_____/\____/\___/\__,_/_//____/\__/\__,_/\___/_/|_|

 💻 LocalStack CLI 0.14.2

[12:15:57] starting LocalStack in host mode 💻
2022-05-02T12:15:57.951:INFO:localstack_ext.bootstrap.licensing: Using test API key
──────────────────────────────────────────── LocalStack Runtime Log (press CTRL-C to quit) ────

LocalStack version: 0.14.2

Starting mock Azure APIs (port 12121)
[2022-05-02 12:16:00 +0200] [63704] [INFO] Running on https://0.0.0.0:12121 (CTRL + C to quit)
2022-05-02T12:16:00.419:INFO:hypercorn.error: Running on https://0.0.0.0:12121 (CTRL + C to quit)
2022-05-02T12:16:00.524:INFO:botocore.credentials: Found credentials in environment variables.
2022-05-02T12:16:00.537:INFO:botocore.credentials: Found credentials in environment variables.
Starting edge router (http port 4566)...
[2022-05-02 12:16:00 +0200] [63704] [INFO] Running on https://127.0.0.1:4566 (CTRL + C to quit)
2022-05-02T12:16:00.739:INFO:hypercorn.error: Running on https://127.0.0.1:4566 (CTRL + C to quit)
```

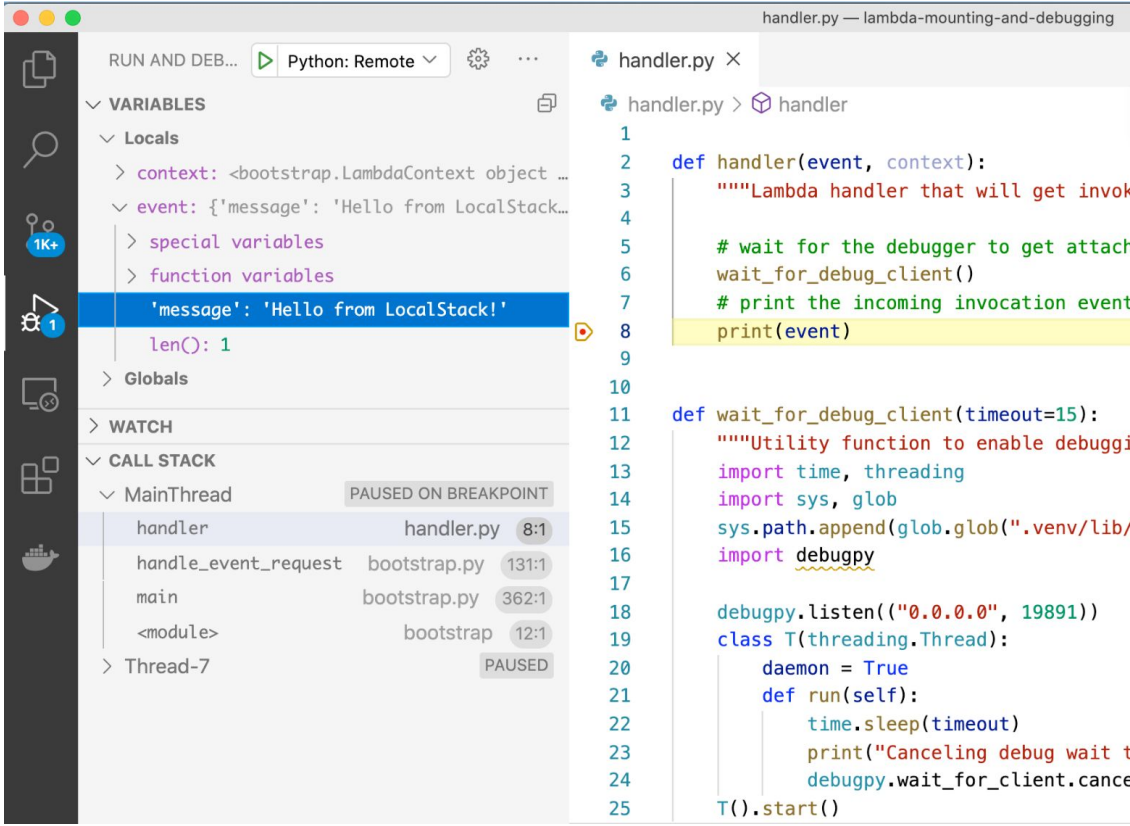
Lambda Hot Reloading

- Default mode for AWS Lambda is to re-deploy the function code
  - Create zip file, upload to cloud, re-create function, ...
- In local dev mode, we can leverage hot reloading
  - User code gets mounted into the Docker container
  - Any changes in the file are immediately reflected in the container
- Enables quick feedback cycles
  - Allows to re-run the Lambda without re-deployment
  - Dumping log/dump files directly to disk for easy debugging



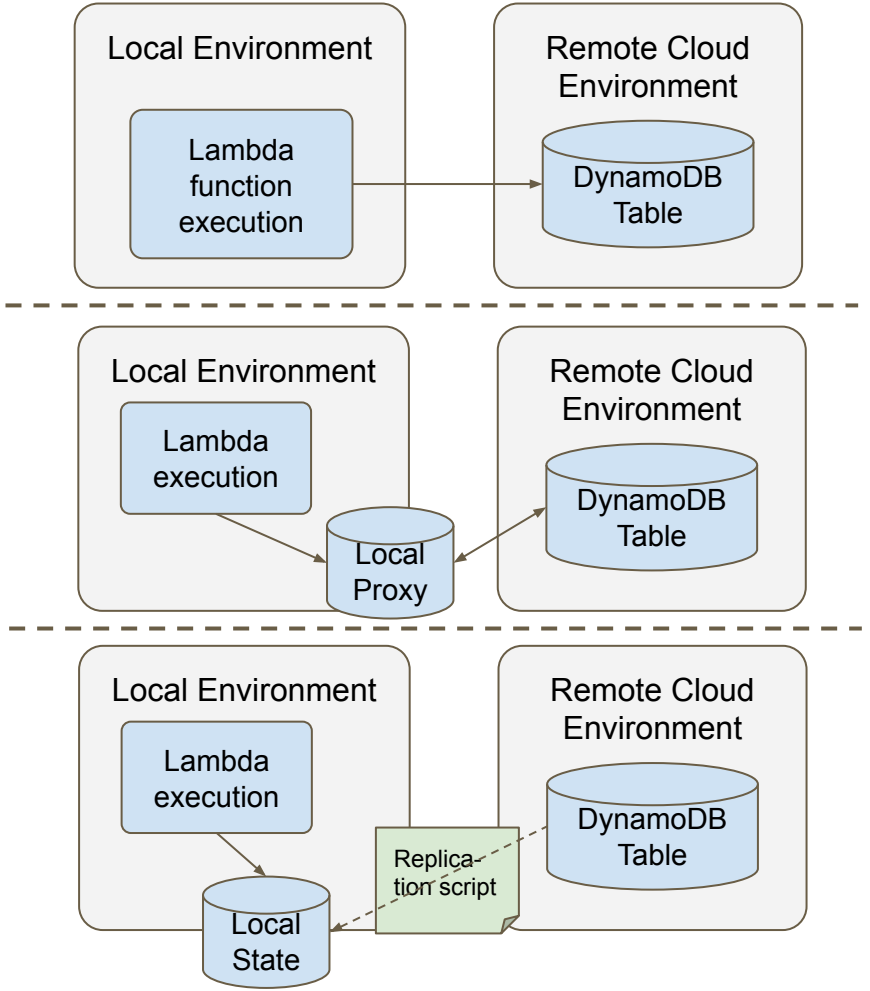
Lambda Debugging





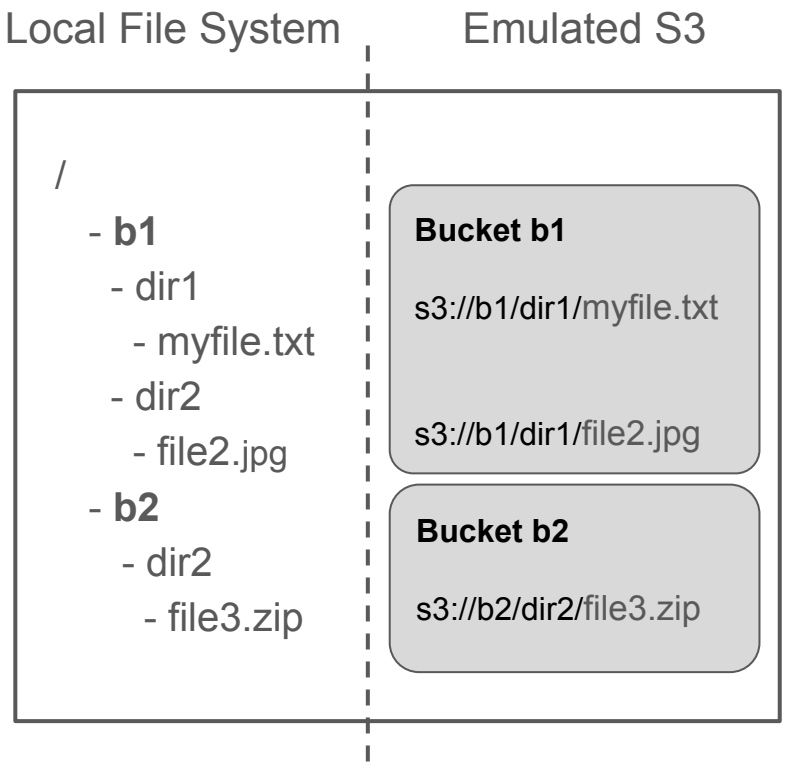
Hybrid Approaches

- Combine real cloud resources with local execution
- Examples:
  1. Local Lambda, remote DB
  2. Outbound / Proxying
  3. Inbound / Replication

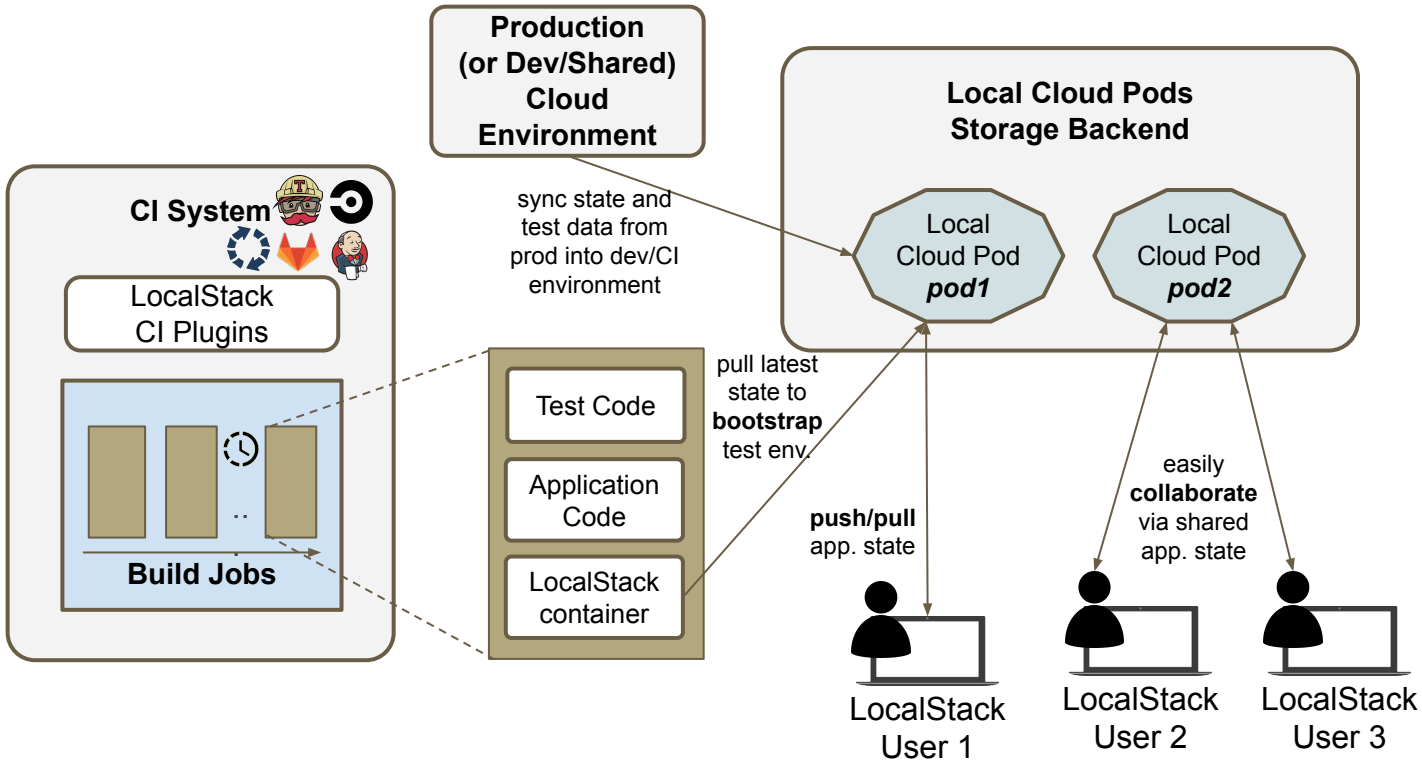


S3 Bucket Mounting

- Ability to run mount local directories as S3 buckets
- Bi-directional mapping
  - parent folders ↔ S3 buckets
  - files ↔ S3 objects
- Useful to s
  - Quickly inspect files locally
  - Observe changes in the FS



Local Cloud Pods - Integration with CI/CD Systems



Conclusion

To conclude ...

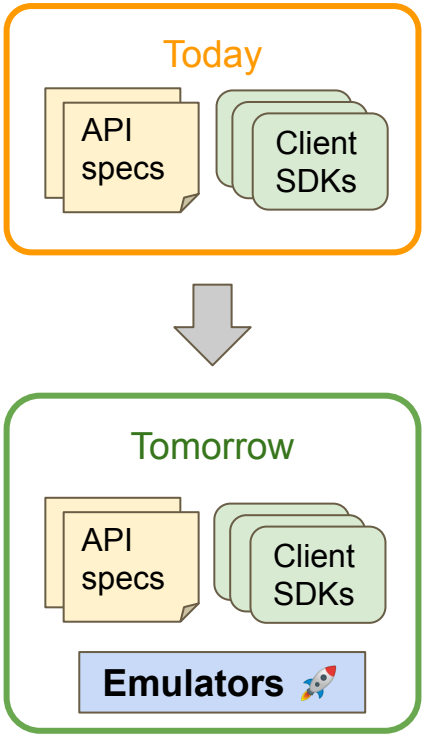
- The Cloud provides economies of scale and unbeatable stability/efficiency for production workloads - however, dev loops sometimes still suboptimal
  - Deploy-test-redeploy loops, local reproducibility, resource conflicts in shared envs, ...
- Different testing methods can be applied (mocking, emulation, hybrid, ...)
  - Also more advanced testing methods like chaos engineering
- Spend some time on researching your ideal cloud testing setup/strategy!
  - Can be an upfront investment, but will pay off over time ...
- Future directions
  - The space of cloud testing/emulation is exciting and evolving fast - give it a try and get involved!

A Vision for the Future ...

- Let's envision a world where:
  - There's an **emulator for every managed service**
  - Emulators becoming table stakes - you "don't exist" if you don't provide one
  - Local dev environments become easily configurable and composable

- Similar to impact of OpenAPI specs on APIs and microservices
  - **API specs** → huge boost for interoperability
  - **Emulators** → huge boost for dev velocity
    - providing a high-fidelity representation of the semantics and inner workings of an API

- It has been demonstrated that it ***is* feasible** (e.g., LocalStack)
  - Hard to imagine a system that is more complex than a cloud platform
  - Yet, emulators can raise the **abstraction** to a level **suitable for local dev**



Try it out - engage with our community

Get started: https://docs.localstack.cloud/get-started/

- Trial of LocalStack Pro: https://app.localstack.cloud/
- Free educational licenses: https://localstack.cloud/educational-license/

Engage with the community:

- Engage with our community and spread the word! Reach out to us with any feedback or to setup joint community events (info@localstack.cloud)

Scan to join our **Slack channel**!

Thank You!

info@localstack.cloud

https://localstack.cloud

@_localstack

https://linkedin.com/company/localstack-cloud

localstack-community