Test-driving the cloud

Agile Prague | Sep 19 2022 | Michal Svoboda

TDD intro/recap

- When tests pass, deploy to production
- Design/architecture is driven by use cases and testability
- Typically, more robust code
- Hard for “infrastructure”

Infrastructure vs cloud app

- Traditional infrastructure – quite static – puppet etc.
- Cloud “infrastructure” isn’t as static
- Cloud apps – need to be programmable and flexible
- TDD is applicable!

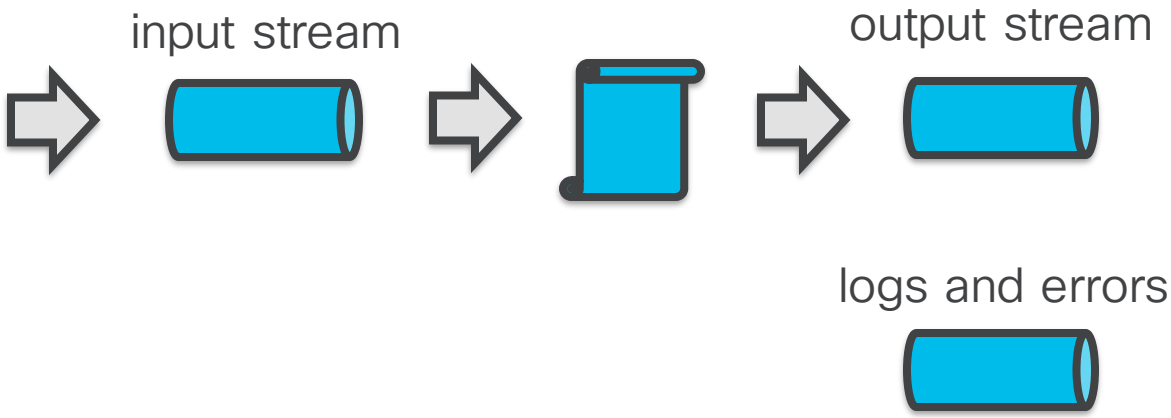
TDD in cloud specific issues

- Slow interactions
- Requires credentials
- Shared testing space (e.g. account)
- Most tooling follows the old “infrastructure mindset”

Cloud app TDD gotchas

- Slow interactions
  - Avoid “big tests”, Incremental deployment
- Requires credentials
  - Separate PROD vs DEV
- Shared testing space (e.g. account)
  - Custom toolkit with “workspaces”
- Most tooling follows the old “infrastructure mindset”
  - Custom toolkit with 1st class callable interface

Demo app



TDD workflow

- Prototype (if needed)
- Write test, deploy app, test fails
- Write app code, test pass
- Refactor
- Anytime when tests pass, deploy to production

Code demo

- End to end tests
- Focused tests

```python
@pytest.fixture(scope="module")
def f():
    fixture = setup()
    fixture.admin_client.deploy_stack(fixture.demo_input_stack)
    fixture.admin_client.deploy_stack(fixture.demo_output_stack)
    access_keys = fixture.admin_client.get_stored_access_key(fixture.demo_output_stack.output_user_name)
    output_client = Client.user(fixture.workspace, access_keys)
    fixture.add_nonpriv_client(output_client=output_client)
    yield fixture


def setup() -> DemoFixture:
    workspace = create_test_workspace()
    demo_input_stack = DemoInputStack.create_for_test(workspace)
    demo_output_stack = DemoOutputStack.create_for_test(workspace)
    admin_client = Client.admin(workspace)
    return DemoFixture(workspace=workspace, demo_input_stack=demo_input_stack,
                       demo_output_stack=demo_output_stack, admin_client=admin_client)


# Functional end-to-end tests


def test_standard_output(f: DemoFixture):
    test_object = create_test_object()
    kinesis_in = f.admin_client.kinesis(f.demo_input_stack.input_stream_name)
    kinesis_out = f.output_client.kinesis(f.demo_output_stack.output_stream_name)
    kinesis_in.put(test_object)
    eventually(EVENTUAL_ASSERT_TIMEOUT_IN_SECONDS,
               lambda: wait_for_test_object(kinesis_out, test_object))


def test_error_output(f: DemoFixture):
    error_message = "Invalid kinesis plain object: not JSON formatted"
    test_object = "b'test'"
    kinesis_in = f.admin_client.kinesis(f.demo_input_stack.input_stream_name)
    kinesis_err = f.output_client.kinesis(f.demo_input_stack.error_stream_name)
    kinesis_in.put_invalid_json('test')
    eventually(EVENTUAL_ASSERT_TIMEOUT_IN_SECONDS,
               lambda: wait_for_lambda_error_message(kinesis_err, error_message, test_object))
```

Code demo

- End to end tests
- Focused tests


```python
# Focused functional and state tests for lambda setup


def test_lambda_invoke_no_errors(f: DemoFixture):
    lmbd = f.admin_client.lmbd(f.demo_output_stack.processing_function_name)
    ret = lmbd.invoke({
        'kinesis': {'data': str(base64.b64encode(b'{"key":"value"}'), 'utf-8')},
        'eventID': 'EVENT_ID',
        'eventSourceARN': 'EVENT_SOURCE_ARN',
    })
    payload = ret['Payload'].read()
    assert payload == b'null'


def test_fan_outs(f: DemoFixture):
    kinesis_in = f.admin_client.kinesis(f.demo_input_stack.input_stream_name)
    consumers = kinesis_in.list_stream_consumers()['Consumers']
    assert match(consumers, m={
        'ConsumerName': f.demo_output_stack.processing_consumer_name, 'ConsumerStatus': 'ACTIVE'})


def test_lambdas_config(f: DemoFixture):
    lmbd = f.admin_client.lmbd(f.demo_output_stack.processing_function_name)
    config = lmbd.configuration()
    assert config['Timeout'] == 360
    assert config['MemorySize'] == 1024
    assert config['Runtime'] == 'python3.9'


def test_lambdas_parallelization_and_buffering(f: DemoFixture):
    lmbd = f.admin_client.lmbd(f.demo_output_stack.processing_function_name)
    assert lmbd.list_event_source_mappings()[0]['BatchSize'] == 500
    assert lmbd.list_event_source_mappings()[0]['MaximumBatchingWindowInSeconds'] == 1
    assert lmbd.list_event_source_mappings()[0]['ParallelizationFactor'] == 4


def test_lambdas_retry_attempts_infinite(f: DemoFixture):
    lmbd = f.admin_client.lmbd(f.demo_output_stack.processing_function_name)
    assert lmbd.list_event_source_mappings()[0]['MaximumRetryAttempts'] == -1


def test_lambdas_event_source_mapping_latest(f: DemoFixture):
    lmbd = f.admin_client.lmbd(f.demo_output_stack.processing_function_name)
    assert lmbd.list_event_source_mappings()[0]['StartingPosition'] == 'LATEST'
```

Experience

- 2 years, multiple projects
- Test everything – function, encryption, scalability, robustness, …
- Massive and aggressive refactoring
- Long term vs short term resources
- Functional vs state tests vs monitoring

The bridge to possible