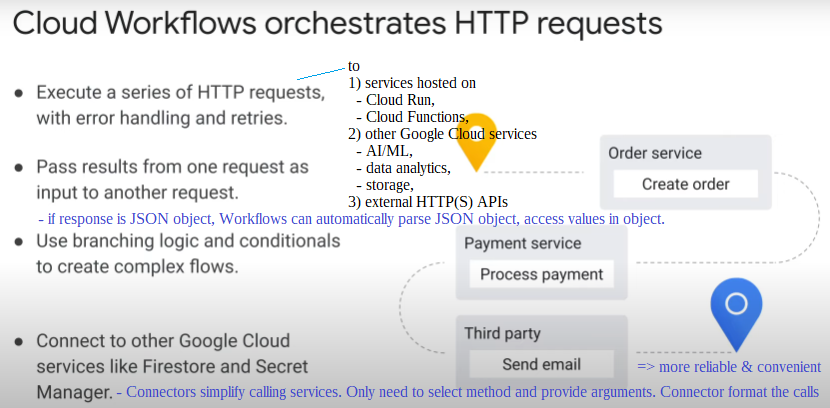
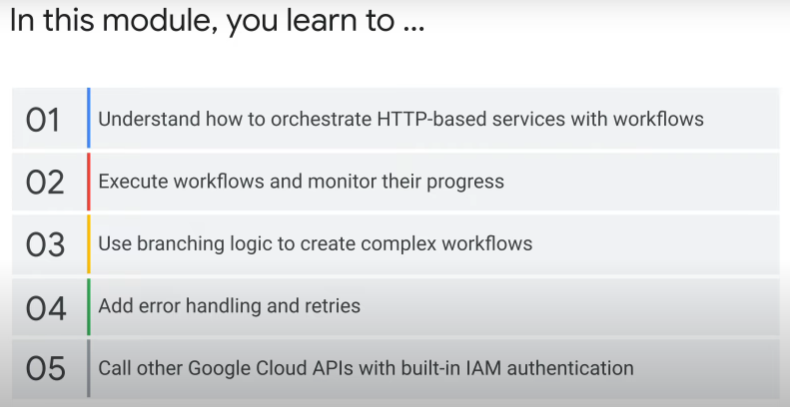
Application Development with Cloud Run

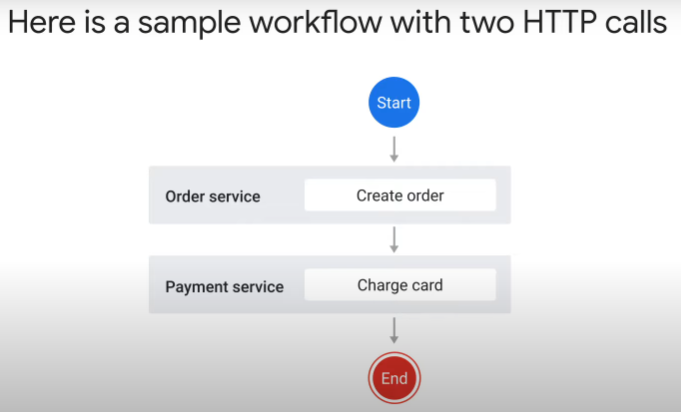
### Building severless Workflows

#### Introduction

* Severless
* Orchestrator
* execute services in a defined order
* use: for application, operational & business processes that span multiple web services
* security: IAM authentication, Identity Aware Proxy.
* Storing Workflows execution state

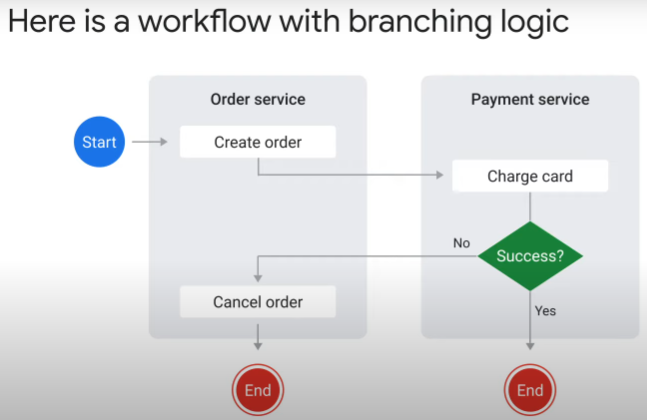
#### Example Workflows

1. Simple linear workflow



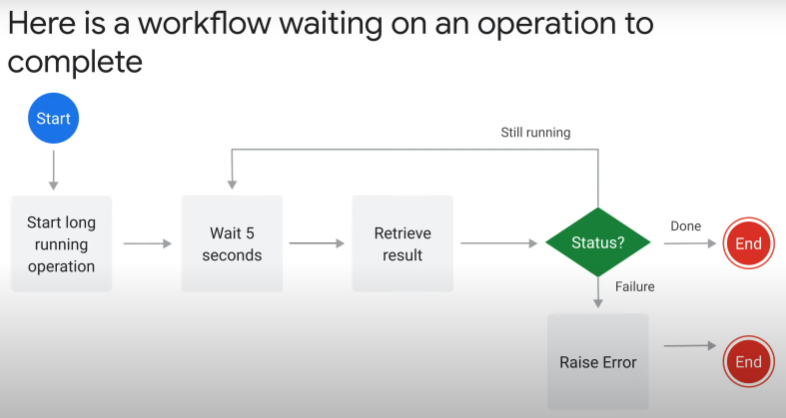
* 2 different Cloud Run services: Order service and Payment service
* Start > create an Order > pass order info to Payment service > charges a credit card

2. Workflow with branching logic



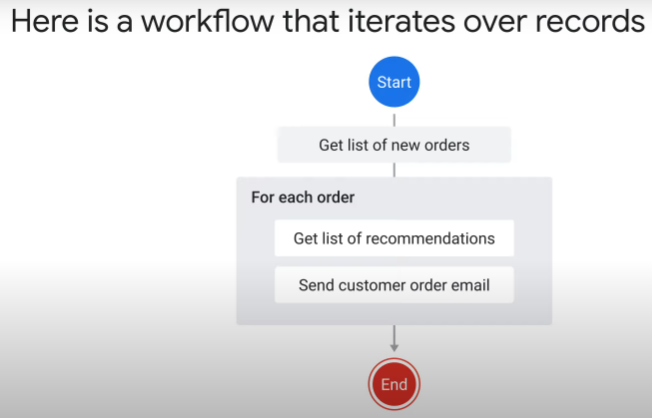
* in real world, an event may have 2 or more outcomes (decision)
* e.g. charge card > can fail or succeed > if fail, cancel order.

3. Workflow that involves waiting



* for Cloud Run services, a HTTP response must be returned within 60 min of the request
  + if not, request ends and error code 504 (gateway timeout) returned
* for Workflows, the max waiting time for an operation before timeout is 1 year.
* Deliberate suspension of execution (sys.sleep)
* poll an API (HTTP GET) until condition met / expected response received
  + long-running operation
  + wait for manual human action

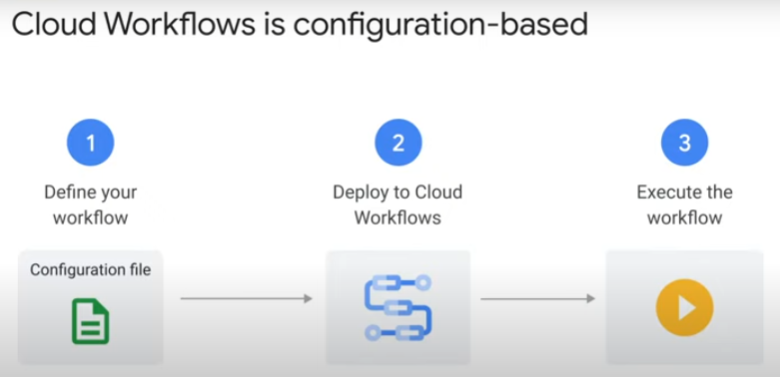
4. Workflow that involves batch processing



* Cloud Run service 1: iterates over entries in a database at a set schedule > get list of new orders by customers
* For each order,
  + Cloud Run service 2: get list of recommendations based on previous orders by that customer
  + Cloud Run service 3: email order confirmation with list of recomended products

#### Workflow development

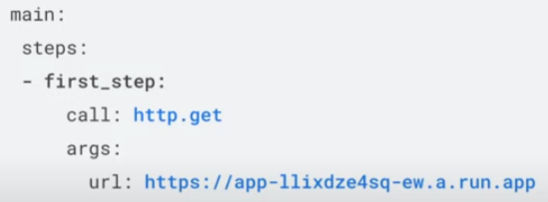
3-step process:



* https://cloud.google.com/workflows/docs/overview

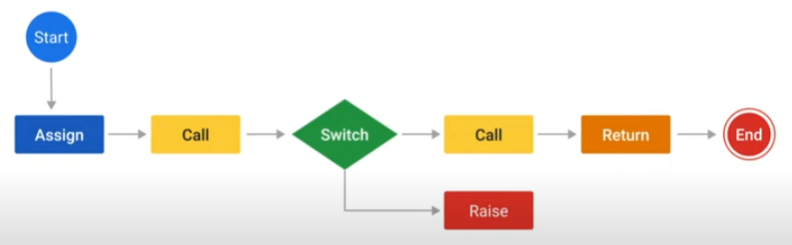
1. Prepare the workflow’s definition [configuration file]

* only 1 main branch
* to define the desired steps and order of execution
* in yaml or json
* syntax: https://cloud.google.com/workflows/docs/reference/syntax
* a minimal example (one step) of a config file in yaml format:



1. Deploy
2. Execute configuration file

Some types of steps that can be found in a workflow:



* “assign” - pass value or result of expression ${ ...} to a variable

- set in the workflow execution memory

- scope sensitive

- https://cloud.google.com/workflows/docs/reference/syntax/variables

* “call” - call a function and store the response returned in a variable in memory

- function can call a connector, a standard library function, or a sub-workflow (user-defined custom function)

- https://cloud.google.com/workflows/docs/reference/syntax/calls

* “switch” - a selection mechanism

- allows the value of an expression to control the flow of execution

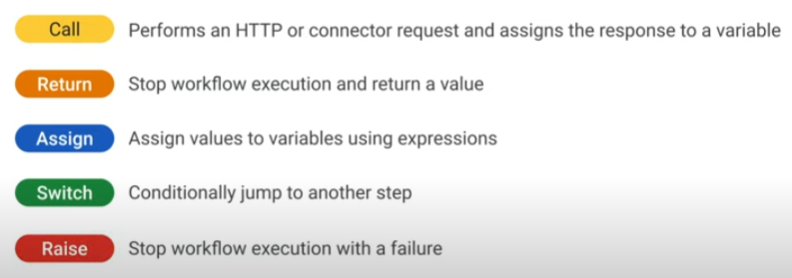
- if value matches, the condition’s statement is executed

- https://cloud.google.com/workflows/docs/reference/syntax/conditions

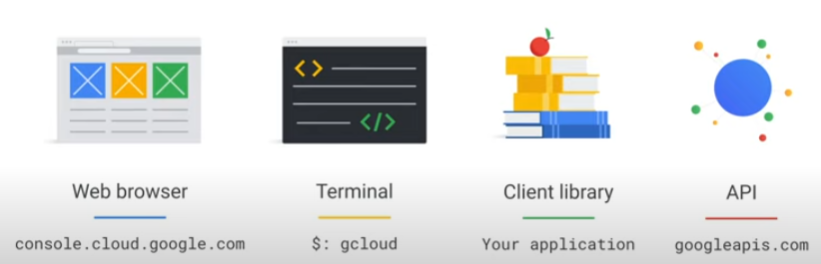
* “raise” - raises custom errors

- ends workflow execution

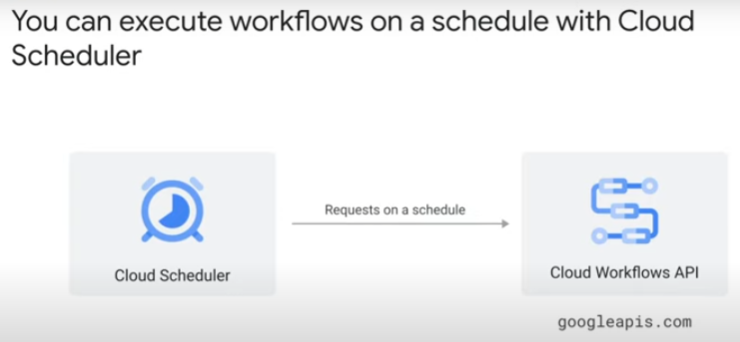
* “return” - ends workflow execution and returns a value or expression
* https://cloud.google.com/workflows/docs/reference/syntax/syntax-search



4 ways to manage and deploy workflows:

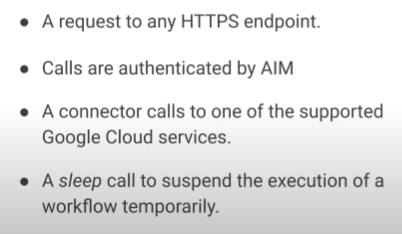


Use Cloud Scheduler to run workflows according to a schedule:

* Cloud Scheduler is severless
* use it to call Cloud Workflows API according to a schedule

#### Call and Return

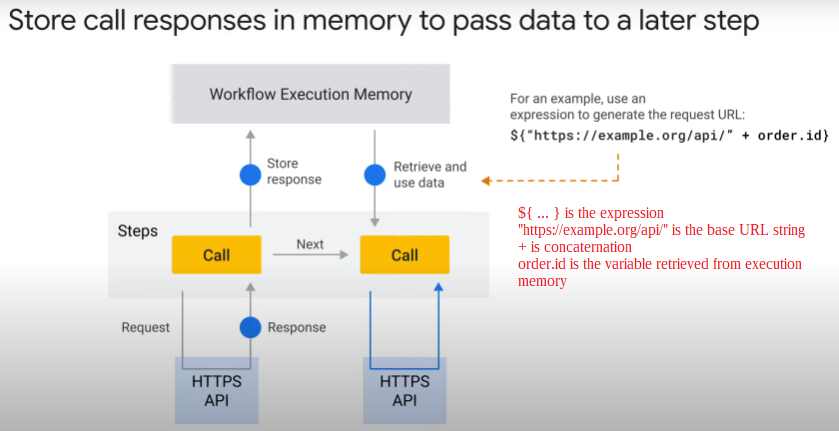
Most of the steps in a workflow are calls:

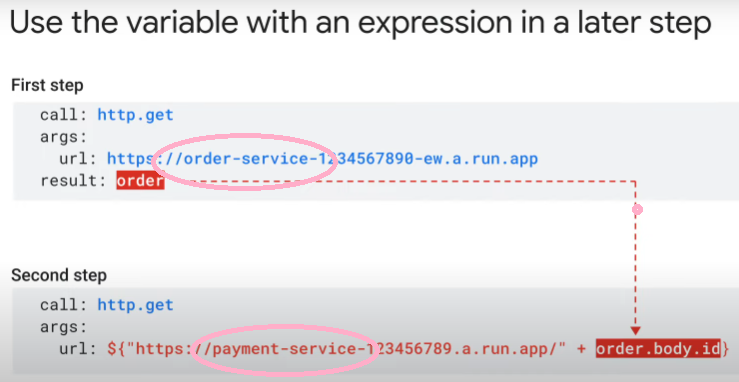


* a call is a request to any HTTPS endpoint
  + can be an external public endpoint
* Workflows support IAM.
  + a call to another Google Cloud Service is authenticated by IAM
  + Cloud Workflows service identity must have the role/permissions for action on that resource
* a sleep call suspends execution of the workflow for a given number of seconds
  + useful when polling for data over given interval
    - e.g. http.get calls at regular interval until API returns jobStatus as complete
  + <https://cloud.google.com/workflows/docs/reference/stdlib/sys/sleep>
  + https://cloud.google.com/workflows/docs/sleeping

Pass data returned in response of one call to the next call:

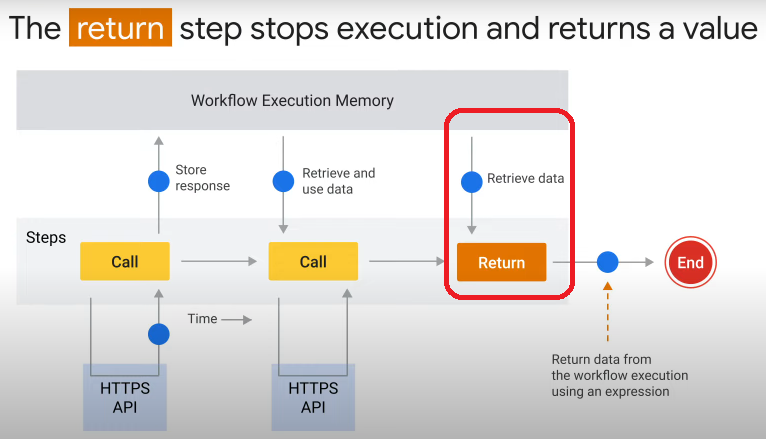
* assign response body returned to an “output variable”.
* variable will be stored in execution memory, and can be retrieved for use in expression to generate URL of next call



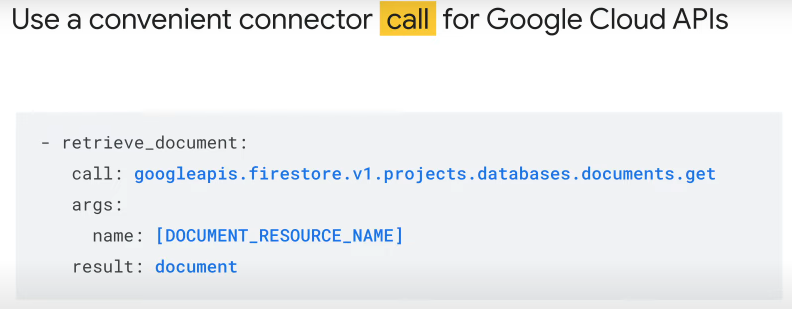
* example: to pass *order.id* from order-service to payment-service
* the definition of this call step:
* Workflows automatically parses .json responses so we can access the value of “order.body.id” in execution memory in the 2nd step.

Need a final return step at end of entire Workflow:

* stops execution and return result of last call (entire Workflow) using an expression
* e,g, return: ${order.body}

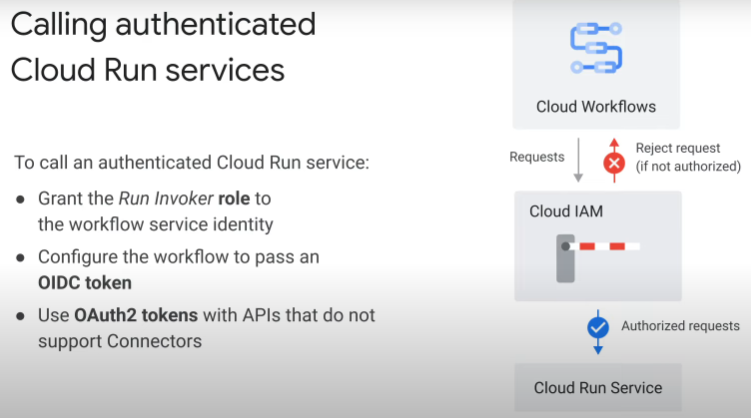


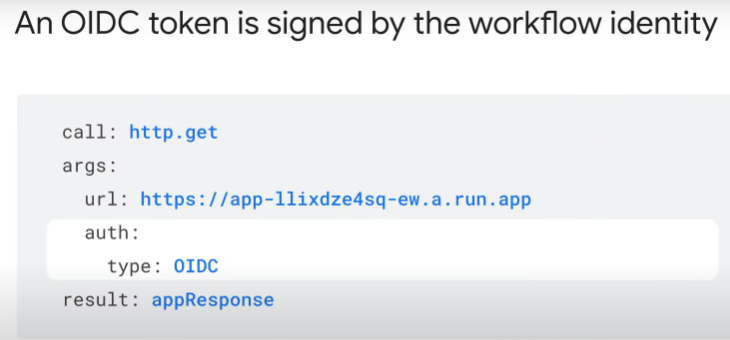
Use **Connectors** to call Google Cloud Services:



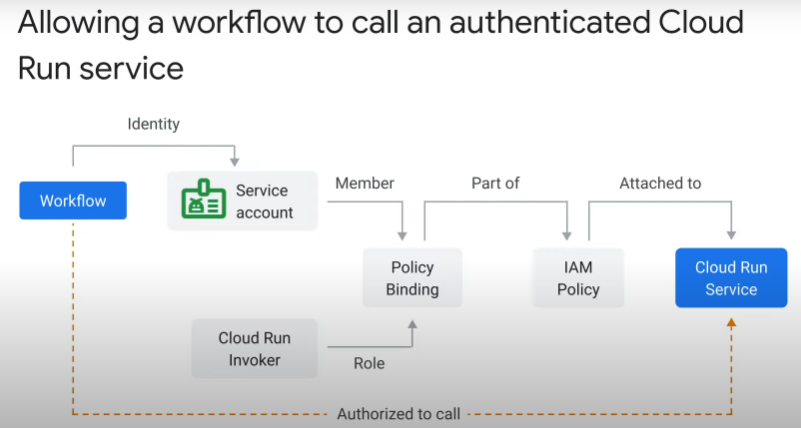
* + [e.g. Firestore – list, store and retrieve documents.](https://cloud.google.com/workflows/docs/connectors)
  + e.g. Secret Manager – load secrets
  + e.g. PubSub – manage topics and subscriptions, publish messages
  + e.g. Cloud SQL admin – create and manage SQL instances
  + reliable functional wrapper around ordinary HTTP calls. Advantages:
    - abstracts away boilerplate codes for
      1. error handling with retries
      * authentication
    1. handles long-running operations transparently without polling
    2. convenient interface to pass parameters
  + https://cloud.google.com/workflows/docs/connectors

Workflows need IAM authentication to call Google Cloud services:

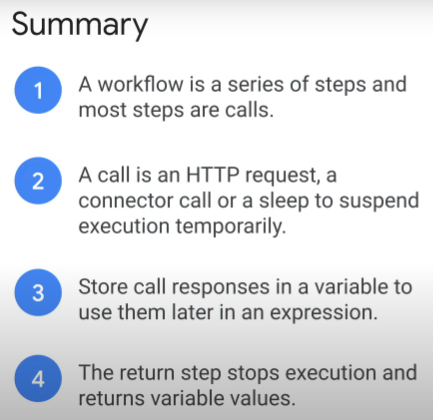
* create Service Account and use it as Workflows' service identity
* example of call definition (part of configuration file) using OIDC token:



Revision: Workflows' service identity and IAM policy



* Workflows has service identity > binds to Cloud Run Invoker role (policy binding).
* Policy binding is part of IAM policy attached to resource (Cloud Run Service)



#### Expressions

Examples of expressions:

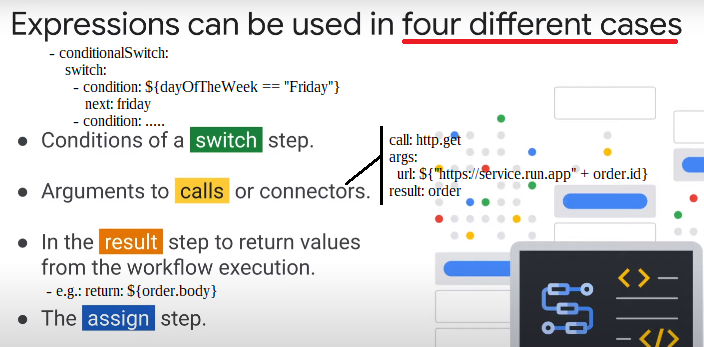
* evaluated by Workflows engine
* output used during execution
* syntax: **${ ... ... ... }**

Note: When call system functions, the **blocking** functions are not supported inside expressions.

* + e.g. sys.sleep and sys.log

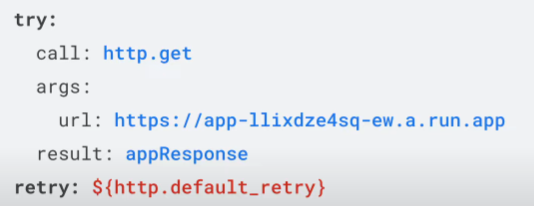
Use case 4: In Assign step

* once an expression is assigned to a variable, use the variable in subsequent steps
* no need to repeat long expressions. Clean codes.

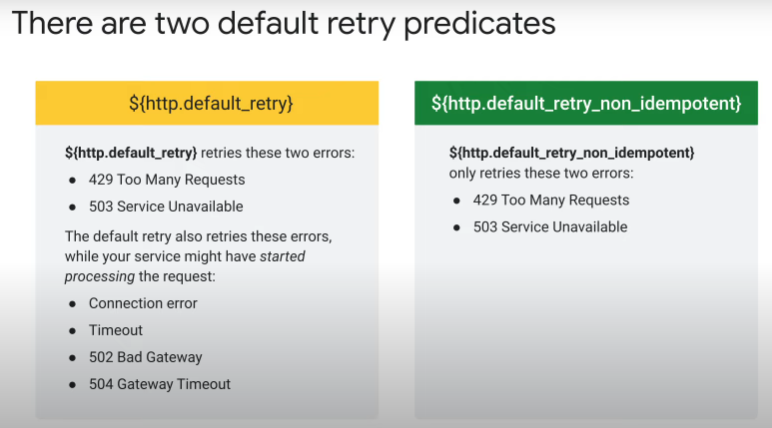


#### Retries

* https://cloud.google.com/workflows/docs/reference/syntax/retrying
* handle errors by retrying
* retry the steps that return specific error codes
* retrying is not a default, need to add a try/retry block/structure



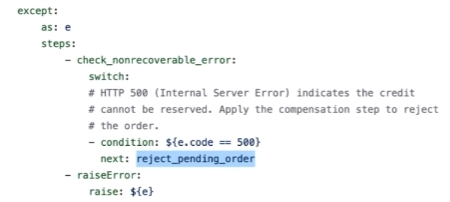
* Workflows has
  + 2 default retry policies
    - for idempotent steps (no effect on output if operations repeated, safe to repeat)
    - for non-idempotent steps (steps cannot be safely repeated without changing output)
  + and custom retry policies
* retry predicate defines which error codes will be retried.
  + defined for the 2 default retry policies
  + for a custom predicate, have to structure as a sub-workflow that returns true if need to retry, false if no need



* ${http.default\_retry}
  + will retry following HTTP status codes, before application begins processing a request
    - 427 when too many requests arrive at application at one time
    - 503 when service unavailable for whatever reason
  + will also retry if encounter following errors while processing a request
    - loss connection > no response code
    - script being executed timeout > no response code
    - 502 Bad Gateway error response code > server acting as the gateway or proxy receives an invalid response from its upstream server
    - 504 Gateway Timeout > server timed out while waiting for another server to respond
* ${http.default\_retry\_non\_idempotent}
  + does not retry errors thrown during processes execution
  + because retrying operations may change output

Retries and saga pattern in distributed systems:

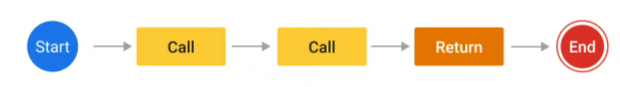
* https://www.youtube.com/watch?v=yqMKr37mGJw
* https://microservices.io/patterns/data/saga.html
* declarative Workflows, avoid coupling, use service orchestrator
* to take care of transient errors, e.g. network issue, service down - wrap call to endpoint in try/retry block with default/custom retry policy
* to take care of permanent error, e.g. always fail when wallet no money
  + retries won’t work
  + use saga pattern
    - a call can have 2 outcomes: success or failure
      * if success > Workflows execution continues
      * if fail > need to take compensatory steps to undo/cancel call
        + how to do in practice: in yaml definition file
        + after try/retry block to handle transient errors, have an except/switch block

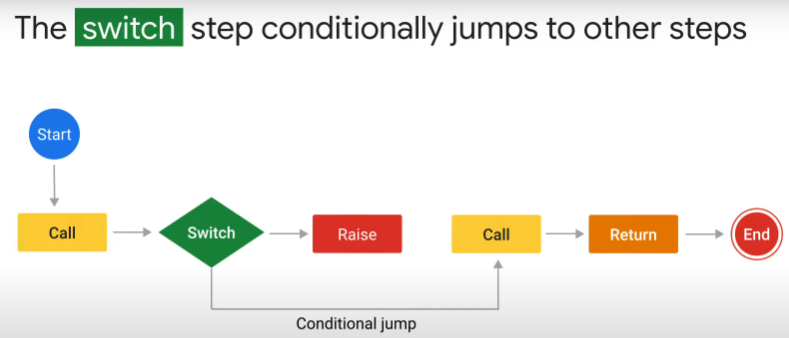


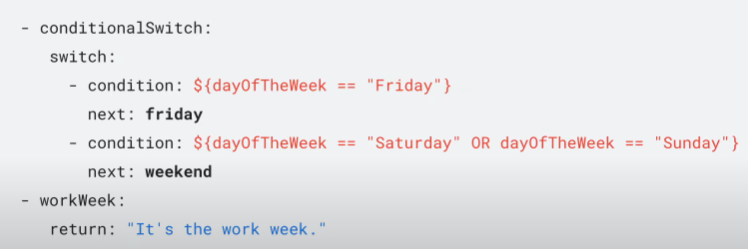
* + in saga pattern, you don’t write to database then delete
  + pattern is eventually consistent

#### Switch and raise

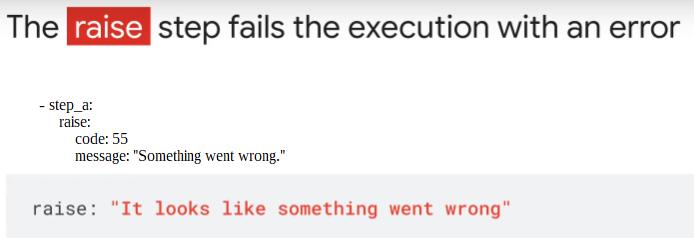
In a linear Workflow, steps are executed in specific order, without branching.

Switch block:



* https://cloud.google.com/workflows/docs/reference/syntax/conditions
* selection mechanism
* let value of expression control flow of execution
* every expression must evaluate to true or false only
* if expression true, that condition’s statement is executed
* all conditions in switch block evaluated in ordered sequence
* if no matching expression, Workflows look for optional default
  + last condition in switch block
* if none of the conditions and default match, Workflows continue execution with next step in code

Raise step:



* stops Workflows with custom error message.
* https://cloud.google.com/workflows/docs/reference/syntax/raising-errors

