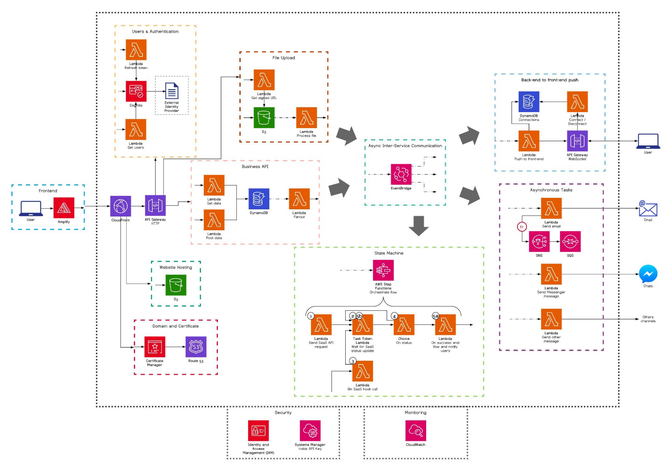
Statically Inferring Usage Bounds for Infrastructure as Code

Feitong Leo Qiao, Mark Santolucito
Barnard College, Columbia University

Motivation

- Accurately estimating the cost of a Pay-as-You-Go IaC deployment is difficult because it requires accurately estimating resource usages.
  - *49% of IT execs find it difficult to get cloud costs under control.**
  - *54% of those believe a primary challenge is a lack of visibility into cloud usage.**
- Historical usage data cannot predict the impact of topological infrastructure changes, and is not even available for new infrastructure.
- It is difficult to know if your estimates of usage bounds for resources are correct within a large infrastructure due to inter-resource interactions.
- Local reasoning may be easy, but global reasoning is hard.

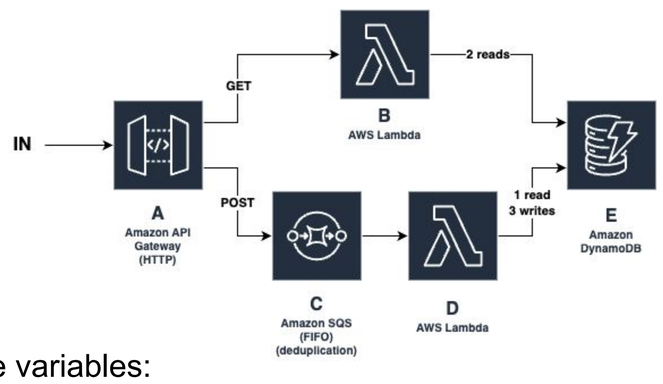


Constraint Generation

Input: IaC definitions files

Output: Resource usage constraints

1) Build directed acyclic graph from IaC definitions, where nodes are resources and edges are direct connections/triggers.



2) For each node, instantiate usage variables:

`A.requests`
`A.GETs`
`A.POSTs`
…
	`B.requests`	`C.requests`	`D.requests`	`E.requests`
				`E.reads`
				`E.writes`

3) For each edge (S->T),
instantiate one edge variable for each of T's usage variables:

`IN_A.requests`
`IN_A.GETs`
`IN_A.POSTs`
…
	`A_B.requests`	`A_C.requests`	`B_E.requests`
			`B_E.reads`
			`B_E.writes`
				`C_D.requests`	`D_E.requests`
					`D_E.reads`
					`D_E.writes`

4) Generate the basic constraints that all variables above should be greater than or equal to 0:

```
A.requests >= 0
A.GETs >= 0
A.POSTs >= 0
B.requests >= 0
C.requests >= 0
…
```

5) Generate constraints at each node. There are three kinds of constraints:

- incoming: constrain a usage variable to the sum of the incoming edge variables
- intrinsic: relate a node's usage variables to each other
- outgoing: relate outgoing edge variables to usage variables

A	`A.requests = IN_A.requests`
	`A.GETs = IN_A.GETs`
	`A.POSTs = IN_A.POSTs`
	`A.requests = A.GETs + A.POSTs`
	`A_B.requests = A.GETs`
	`A_C.requests = A.POSTs`

B	`B.requests = A_B.requests` `B_E.reads = 2 * B.requests`

C	`C.requests = A_C.requests`
	`C_D.requests <= C.requests`
	`(C.requests > 0) -> (C_D.requests > 0)`

D	`D.requests = C_D.requests`
	`D_E.reads = D.requests`
	`D_E.writes = 3 * D.requests`

E	`E.reads = B_E.reads + D_E.reads`
	`E.writes = B_E.writes + D_E.writes`
	`E.requests = E.reads + E.writes`

Denote the conjunction of all constraints above as **K**, to be used in **Use Cases**.

Use Cases

Bounds on single resource usage

*"If we receive 100 user requests,
how many read requests will the DynamoDB (**E**) have?"*

1) Constraint for user-provided usage estimate, **U** = `IN_A.requests = 100`
2) Form SMT formula, **P = K ∧ U**
3) Form optimization objective, **O** = `E.reads`
4) Invoke an optimizing SMT solver on **P** with objective **O** twice:
   - The 1st time: minimize the objective -> yields `1` *(lower bound)*
   - The 2nd time: maximize the objective -> yields `200` *(upper bound)*

Bounds on whole infrastructure

*"If we receive 100 user requests,
what are the bounds on the cost of my entire IaC stack?"*

The bounds may be computed using the same process as above, but with a price function as the objective:

O = `price( A.requests, B.requests, C.requests, D.requests, E.requests, E.reads, E.writes )`

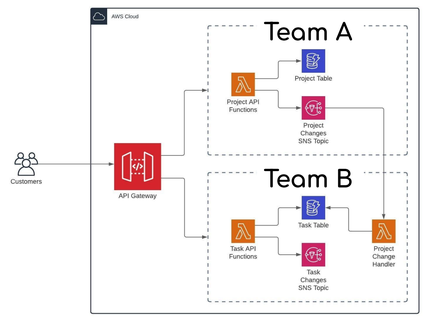
Compositional reasoning about usage across infrastructure stacks

"How do Team A's infrastructure changes affect Team B's cloud spending?"

The bounds on the cost of Team B's infrastructure may be computed using the same process as above, but with the the cost of Team B's infrastructure as the objective:

O = `price(` *`Team B's resource usages`* `)`

This allows more fine-grained understanding of how some infrastructure changes could affect another component of the system.



Validating usage estimates

"Are my usage estimates wrong?"

Suppose a user estimated the cost of an IaC stack, and provided usage estimates for each resource:

`A.requests = 100, E.reads = 300, …`

1) Denote the conjunction of the equalities above as **U**.
2) Form SMT formula, **P = K ∧ U**
3) Invoke SMT solver on **P**. If the result is *UNSAT*, it means that user-provided estimates are wrong. Can explain with unsat core.

Future Work

Application-level constraints

Constraints for services with programmable behaviors (e.g. AWS Lambda) are not discoverable at the IaC configuration level. To generate constraints for such services, we need to perform application-level analysis, such as static program analysis or log analysis.

*"2022 State of Cloud Cost Report". Anodot, 2022.