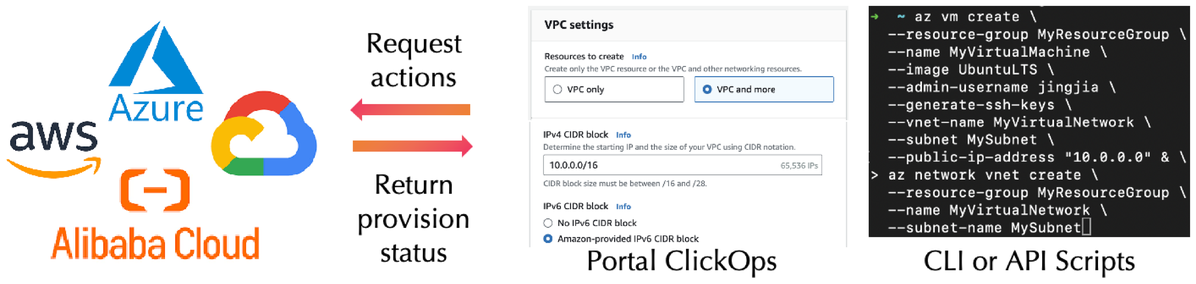
LILAC: Automated Program Lifting for Cloud Infrastructure-as-Code

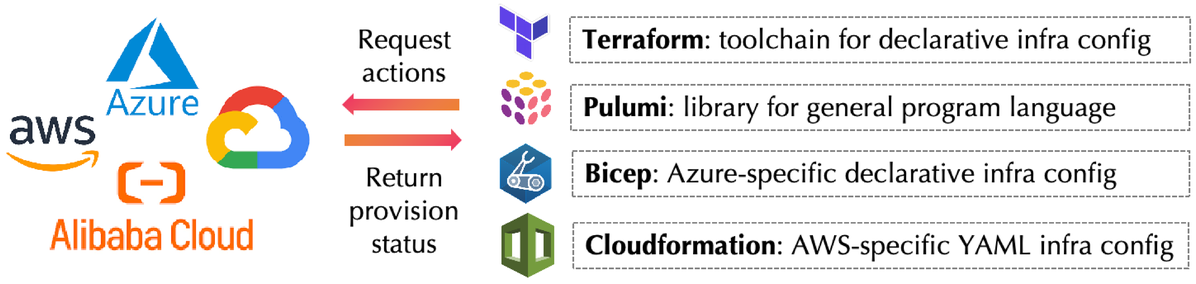
Presented by: Jingjia Peng Advised by: Ang Chen, Xinyu Wang
Mentored by: Yiming Qiu, Patrick Tser Jern Kon, Yibo Huang, Zheng Guo, Pinhan Zhao

Cloud Management and Infra-as-Code

Cloud resources are difficult to configure and manage. Previously, cloud tenants have to employ extensive cloud/DevOps engineering teams to handle obscure cloud service interfaces. Those interface or cloud management tools include Command Line Interface (CLI), cloud API script and ClickOps on cloud website portals. However, these tools are reckoned extremely **inefficient and unreliable when managing large-scale production environments.**

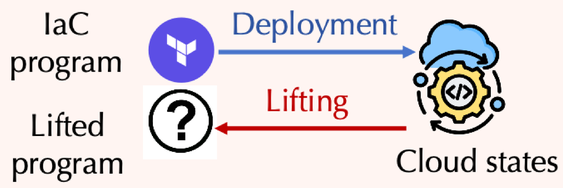


"Infrastructure as Code" (IaC) paradigm arose to simplify cloud management. Cloud IaC enables users to write only high-level code to express their intended infrastructure, **shielding them from low-level details about how the underlying infrastructure is deployed.** Among various mainstream IaC platforms, ***Terraform*** is the most popular one and also our initial target in Lilac.



Lifting Cloud Infrastructure-as-Code

In **IaC deployment**, IaC compiler automatically calculates the difference between our desire infra and current infra and evokes a sequence of cloud API calls to modify our cloud resources.



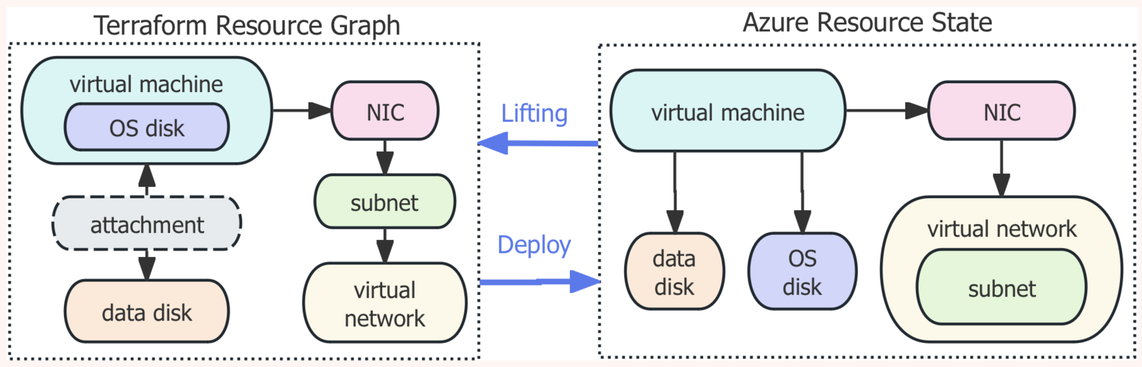
However, **IaC lifting, synthesis of IaC program based on cloud state of existing infra**, is poorly supported. Lifting is crucial for cloud DevOps when we need to bring existing resource, or "brownfield infra", back to the IaC management plan, which are historically created by other tools (CLI, API script or ClickOps). **Extensive industrial efforts** have been invested to address IaC lifting demand, but their outcomes are far from satisfaction.

IaC platform attempt	Third-party developers attempts
Terraform Import	TerraCognita by cycloid
✅ support all Terraform resources	✅ resource-specific optimization
❌ error-prone guesswork	❌ based on handcoded rules, still error-prone
❌ manually write config blocks	❌ only support some popular resources

Challenges Towards Automated IaC Lifting

Step 1 **topology lifting**: we first figure out ***what IaC resources we need in our desired program***, according to the matching between IaC resource blocks and cloud resource entities.

- Challenge 1: Cloud resource graph is obscure to obtain, due to unclear definition or boundary of some resources, like nested sub-resource (subnet) and connection as resource (disk attachment).
- Challenge 2: There is no one-to-one mapping between IaC and cloud resource graph. An **asymmetric mapping** example is as followed.



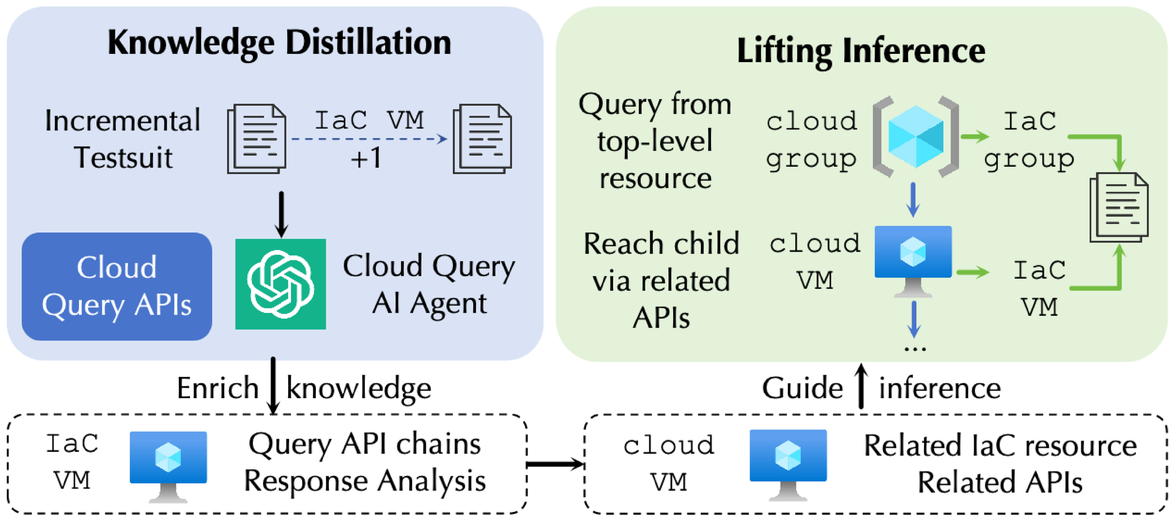
Although manually encoding all mapping rules seems possible, but the **complexity and diversity of cloud resources** make it impractical.

Step 2 **attribute lifting**: we pop up attribute values for each IaC resource block. Although in different form, those values must be equivalent to those in the cloud configuration while conforming IaC syntax rules.

We currently focus on step1, whereas step 2 will be integrated later.

LILAC Topology Lifting

- **Knowledge Distillaion**: We deploy incremental testsuit, where the only difference between two sequential test is the addition of one IaC resource. For each added resource, we can obtain the necessary rules to fully capture its cloud counterpart with the aid of AI agent.
- **Lifting Inference**: In brownfield deployment, we first query from the top-level resource such as resource group, and query its child resource by related APIs. Knowledge base helps us map observations in the query chain back to IaC counterparts.



Email: jingjia@umich.edu