How Can Infrastructure as Code Accelerate Data Center Bring-ups? A Case Study at ByteDance

Xianhao Jin*, Yifei Feng*, Yufei Gao†, Yongning Hu*, Jie Huang*, Kun Xia*, Luchuan Guo*
ByteDance Ltd.
*San Jose, CA, USA †Seattle, WA, USA
{xianhao.jin, yifei.feng, yufei.gao, yongning.hu, jie.huang, kun.xia, luchuanguo}@bytedance.com

***Abstract*—Software companies are establishing new data centers to enhance software performance with lower response times as well as meet security requirements for storing local user data. ByteDance also has a strong need to deploy its services to new data centers worldwide quickly and with minimal error. Unfortunately, this process can also be time-consuming and error-prone, *e.g.,* services often have dependencies on one another, requiring a strict deployment execution order. Moreover, the high similarity between resources in the new and the old data centers enables minimal modifications and maximizes the reuse of existing infrastructure configurations while providing the ability of global resource management. Additionally, manual migration across data centers often requires multiple confirmation checkpoints, which can significantly slow down the process. Therefore, to accelerate the new data center bring-ups, we adopt the idea of Infrastructure as code (IaC) which is a practice to automatically configure system dependencies and to provision local and remote instances [41]. In this work, we propose ByteRollout, an automatic intent-based software resource deployment system that is able to take customized infrastructure configurations as input and actuate deployments accordingly. We also assess ByteRollout in the following events of new data centers creations driven by site reliability engineering (SRE) teams. The evaluation results demonstrate that ByteRollout significantly accelerates the data center bring-up process, saving months of human effort and reducing costs by millions of USD simultaneously. The results also highlight that the Infrastructure as Code practice can be leveraged in the context of data center setups, offering benefits such as reduced process time and minimized errors throughout the deployment.**

***Index Terms*—infrastructure as code, heterogeneous data centers, software maintenance, empirical software engineering**

I. Introduction

Heterogeneous data centers, with a variety of compute, memory, and network resources, are becoming increasingly popular to address the resource requirements of time-sensitive applications [3]. Heterogeneous infrastructures offer on-demand self-service, broad network access, resource pooling, and rapid elasticity [34]. Despite these benefits, such an infrastructure poses many challenges related to security, deployment, and resource management [21]. As the owner of several globally popular software applications, ByteDance has a strong need to deploy its services and resources across various local data centers to tackle challenges related to compliance, growth, and disaster recovery. Consequently, seven new data centers were built in 2023, a substantial increase compared to the previous practice of constructing only one data center per year. However, based on past experience, this process can be time-consuming and error-prone – deploying new services involves a series of transactional tasks, such as multiple registrations, which require significant human involvement and ultimately hinder efficiency. Moreover, dependency relationships between different services are common and can further increase the risk of failure in manual deployments.

To address the aforementioned issues and facilitate efficient data center bring-ups, we explored various potential solutions and ultimately decided to adopt the concept of Infrastructure as Code. Infrastructure as Code (IaC) is an infrastructure automation approach inspired by software development practices, emphasizing consistent and repeatable routines for provisioning and modifying systems and their configurations, where changes are made to definitions and deployed through unattended processes that incorporate thorough validation [35]. Implementing Infrastructure as Code in our scenario offers three significant advantages: (1) utilizing source-of-truth configuration files reduces human effort, as users only need to add a single line to the placement field of the resource to incorporate a new data center; (2) storing these configuration files within the codebase allows for benefits such as version control and code review; (3) the centralized nature of these configuration enables the management of resources from a global perspective.

In this work, our goal is to develop an automatic platform that enables users to seamlessly copy and paste their resources during a new data center bring-up. To achieve this goal, we propose ByteRollout by taking advantage of the concept of Infrastructure as Code [36] and utilize configuration files as the definitive source to describe the intent of the deployment. Although existing open-source tools such as Terraform [8] are available, we chose to develop our own in-house platform to better accommodate the specificity and diversity of ByteDance's resources while ensuring the system's scalability. A typical happy path for our users in this scenario involves translating an existing online resource into a configuration file, making minor modifications by adding the placement of the new data center, submitting the configuration file to the codebase for code review, triggering the pipeline engine to initiate the deployment after the code is merged, and querying the deployment progress. On our server side, the designed workflow reads the content of configuration files from the codebase, constructs them into a deployment, splits the deploy-

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00256

ment based on different global regions, and concurrently sends the deployment requests to the corresponding regional resource orchestration services. The product must also meet the requirements for rollback, cancellation, query dashboard, dry-run diff confirmation, and dependency relationships. Therefore, we design BYTEROLLOUT as an automated resource deployment platform that provides capabilities for actuating, dry-running, canceling, rolling back, and querying deployments.

After its completion, BYTEROLLOUT is used to deploy resources during the bring-ups of several new data centers for its business products, where we also had chances to evaluate BYTEROLLOUT in terms of cost saving. In this paper, we are highlighting the evaluation results in three data centers bring-ups. In the first data center bring-up, BYTEROLLOUT was able to save over 96% of human effort (measured in "developer days") and reduce the overall process time by 70%. During the bring-up of the second data center, BYTEROLLOUT managed to reduce the overall duration by 50% and the deployment phase duration by over 60%. BYTEROLLOUT also saved over 400 "developer days" of human effort as well as millions of dollars in total throughout the procedure. In the third data center bring-up, BYTEROLLOUT was able to save over 500 developer days of human effort. During the last two data centers bring-ups, BYTEROLLOUT was able to deliver the deployment with a 100% success rate, significantly reducing the amount of time spent on bug fixes. The evaluation results show that BYTEROLLOUT can significantly accelerate data center bring-ups, minimize errors throughout the process, and ultimately save the company a substantial amount of money. This also demonstrates that Infrastructure as Code can enhance the data center setup process by reducing developer effort and minimizing machine waiting time.

II. BACKGROUND

A. Heterogeneous Data Centers

Enterprises traditionally owned proprietary infrastructures for storage and computing, but in the past decade, many have shifted to a heterogeneous setup combining their own resources with those from public or private cloud providers [4]. Heterogeneous data centers use a combination of different server types, such as those with CPUs and GPUs, to enhance performance across various workloads, providing greater flexibility and efficiency. Nowadays, due to the requirements of political policies and concerns over local user data security, heterogeneous data centers are also commonly used to satisfy regulatory compliance, ensure data sovereignty, and enhance the protection of sensitive information.

B. Pipeline Engine for Continuous Delivery and Continuous Deployment

Continuous Delivery (CDE) is a software engineering approach in which teams continue to produce valuable software in short cycles and ensure that the software can be reliably released at any time [10]. Companies that practice continuous delivery have reported great benefits, such as significant improvements in time to market, customer satisfaction, product quality, release reliability, productivity and efficiency, and the ability to build the right product through rapid experiments [11], [29]. As best practice, continuous delivery often works in conjunction with Continuous Integration (CI) to automate the process of establishing the infrastructure and releasing applications. Another concept similar to Continuous Delivery is Continuous Deployment (CD), which refers to the automatic and continuously deploy of the application to target environments. What differentiates continuous deployment from continuous delivery is a production environment (*i.e.,* actual customers): the goal of continuous deployment practice is to automatically and steadily deploy every change in the production environment [50]. Continuous deployment often involves canary deployment for testing in the production and multiple control plane deployments for better performance, high availability, and high reliability.

ByteDance's in-house pipeline engine, used to execute the workflows for Continuous Delivery and Continuous Deployment, is called ByteCycle Pipeline [22]. It includes numerous public atoms that can be combined to build the pipeline within the company. These atoms offer specific, useful features for Continuous Delivery, such as canary deployment and health monitoring, enhancing the overall pipeline's functionality and reliability. In the design of BYTEROLLOUT, we leveraged this pipeline engine to manage the overall workflow, breaking the process into several steps, including deployment generation, dry-run diff confirmation, and query deployment status, *i.e.,* BYTEROLLOUT developed its own atoms that can be publicly used by all pipelines. The integration of the pipeline engine enables BYTEROLLOUT to efficiently manage its workflow and provides an initial user interface for better interaction.

C. Resources at ByteDance

To meet the diverse needs of development activities, ByteDance utilizes multiple categories of technologies that are commonly used across the organization, such as RPC (Remote Procedure Call) services, relational databases, non-relational databases, and traffic governance systems. To better manage the instances of these technologies across different control planes in our system, we abstract them as resources that BYTEROLLOUT is designed to provision. Below, we introduce the essential resources supported by BYTEROLLOUT:

1) TCE services and clusters: Toutiao Cloud Engine (TCE) is a platform primarily designed to manage RPC services and provide users with fast and efficient service deployment solutions. It focuses on service lifecycle management, including tasks such as construction, upgrades, and rollbacks, while striving to deliver highly available and elastic container services. A **TCE Service** defines the logical concept of a service. It encompasses elements such as the service name, unique service identifier (PSM), code repository, image version, startup scripts, and release permission control. It corresponds to a node in the service tree and facilitates integration with other development platforms (e.g., quota management, alert management, and traffic governance).

A **TCE Cluster** is a deployment unit for a TCE Service and can have its own version, instance configuration, instance count, environment variables, and other related information. It consists of instances that directly handle traffic, with the cluster name serving as the smallest naming unit in the service discovery system. Based on a TCE Service, a TCE Cluster includes deployment-related details such as resource packages, instance count, data center, environment variables, and more. The TCE Cluster concept allows for request separation (similar to implementing lanes manually); for example, a public service in the middle layer can provide support to different apps through different clusters. For better management, we abstract TCE Services and Clusters as two separate types of resources. A TCE Service may contain zero or more TCE Clusters, and each TCE Cluster should be deployed within a single Virtual Data Center (VDC). Given that a TCE Cluster cannot exist independently without a TCE Service, there is an inherent dependency relationship between TCE Clusters and Services. TCE is the primary resource at ByteDance, and as such, we primarily use it as the example throughout this paper.

2) Other resources: BYTEROLLOUT also supports other resources such as TCC, RDS and Neptune. Toutiao Config Center (**TCC**) is a configuration management solution provided to businesses, consisting of a platform and SDK (Software Development Kit). It includes features such as configuration management, version management, multi-region and multi-environment support, permissions management, and grayscale release. The online services can access TCC to read configurations and incorporate these configurations into their business logic code. BYTEROLLOUT supports this resource and allows users to modify and deploy their changes to the configuration contents. The Relational Database Service (**RDS**) is a cloud-based service that provides access to a relational database management system (RDBMS) such as Oracle, MySQL, or Microsoft SQL (Structured Query Languag) Server. It allows users to create, manage, and scale relational databases without the need to install or maintain the underlying infrastructure. Relational database services typically provide features such as automatic backups, point-in-time recovery, and multi-zone replication for improved availability and durability. The BYTEROLLOUT feature supports this resource by enabling users to modify and deploy changes to database schema elements, such as table definitions and primary keys, while preserving the integrity of the stored data. The **Neptune** platform provides multidimensional and multi-scenario traffic governance capabilities, helping users improve testing and operation efficiency, as well as the security, stability, and availability of services and systems. The Neptune platform includes the following core modules: security management, scheduling management, stability management, quota limiting, lane control center, and administrator dashboard. BYTEROLLOUT supports Neptune as a resource to allow users to set up the rate limiting, retry configurations, traffic scheduling, timeout and security governance after the TCE services and clusters are successfully deployed. A typical deployment sequence for the mentioned resources begins with ensuring that RDS and TCC are set up before deploying TCE. The deployment of Neptune resources will commence once the TCE clusters are prepared to receive traffic.

D. Regional Resource Orchestration Service (rROS)

At ByteDance, as the initial practise of IaC, developers and SREs can create templates following the specifications defined by the regional Resource Orchestration Service (rROS) to declaratively define the required cloud resources, such as TCE Services and Clusters, along with their interdependencies. The orchestration engine then automates the creation and configuration of these resources based on the template, enabling fully automated cloud resource management. As a result, users only need to maintain a single template to manage infrastructure settings, embodying the Infrastructure as Code paradigm. However, rROS has certain limitations. First, it was not designed with global deployment in mind, which was reasonable at the time when most business teams operated primarily within the China control plane. As a result, rROS continues to provide only regional specifications, requiring users who need global specifications to construct them manually. Second, rROS offers a user interface for interaction, storing configuration files on its platform rather than in the codebase. This adds complexity, as users must track their configurations across multiple platforms, further fragmenting configuration management. To overcome these challenges, BYTEROLLOUT is built on top of rROS, utilizing it as the lower-level regional interactor with resource providers.

III. THE LANDSCAPE BEFORE BYTEROLLOUT

In this section, we outline the essential steps for setting up a new data center without using BYTEROLLOUT in Figure 1 and examine the challenges associated with this approach.

At ByteDance, the process of bringing up a data center typically begins with preliminary research and sign-off, ensuring that all involved teams are aligned before officially kicking off the setup. This involves assessing requirements, aligning with key stakeholders, and securing approvals. Once the initial alignment is complete, the process moves to supply chain delivery, where essential infrastructure components are purchased and delivered. This is followed by infrastructure and system setup, ensuring that the necessary hardware and software foundations are in place. Subsequent steps include batch data batch migration from other data centers and addressing data split-brain issues, ensuring seamless data consistency across systems. The R&D infra and system deployment are launched simultaneously with traffic deployment to begin handling workloads. Both steps are followed by a business access evaluation to verify that the system operates as expected.

Besides, the compliance team will also ensure the components of compliance are deployed successfully, followed by an evaluation from the business team after the micro-services are successfully executed to fulfill the collected business requirement information in the new data center. Before full-scale business operations, the setup undergoes quality assurance (QA) acceptance, A/B testing, and metrics evaluation to ensure

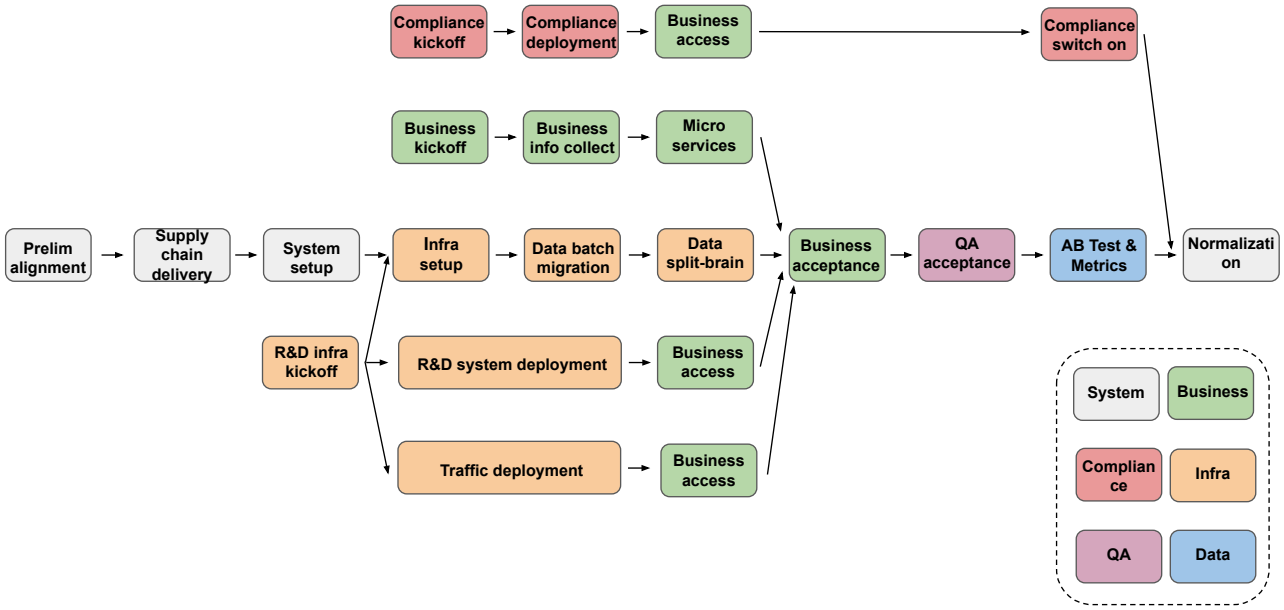

Fig. 1: Essential steps for setting up a new data center before using ByteRollout.

system stability. Finally, the process concludes with compliance switch-on, business normalization, and full operational readiness of the data center. This process requires collaboration among multiple teams, including system, infrastructure, compliance, data, and QA teams, along with the relevant business development teams. ByteRollout is designed to assist the business development team with the phase of deploying micro services because the preliminary analysis reflects that this process is time-consuming and error-prone.

This procedure has several weaknesses, and Infrastructure as Code can address some of them to varying degrees. First of all, the deployment of the infrastructure platform as a foundational software layer takes a considerable amount of time, which poses a similar challenge for the business development team. This issue cannot be resolved by IaC, as the infrastructure platform, being a lower-level platform, is not ready yet. Second, determining the scope of the resources to be deployed and their corresponding deployment methods can be challenging. IaC can address this issue by storing the resources' metadata as configuration files in the codebase, making them easily accessible. Third, data migration is time-consuming due to the need for compliance checks, which add an extra processing time to the process. The batch deployment of services and related resources can take a long time, as the process is prone to failure if dependent resources are not ready. However, ByteRollout can address this issue by providing a dependency-based deployment approach and automatically retrying the deployment if it fails. Moreover, since ByteRollout relies on configuration files as the source of truth and requires minimal modifications during deployment, the success rate can be significantly improved. This, in turn, reduces the time spent on debugging and metric evaluation phases. Considering that bug fixes involve collaboration across multiple teams, which increases communication costs, a higher deployment success rate can save more time than estimated. Finally, once the services are successfully started, the SRE teams need to increase the capacity of the clusters, which also takes time. With ByteRollout, this process can be automated, making it more efficient and faster.

IV. The Design of ByteRollout

We designed ByteRollout to take a configuration file as the source of truth (SoT), containing all the necessary information for deploying resources based on the intended specifications. Users can begin by importing online resources into SoT files, a feature provided by ByteRollout. ByteRollout consists of four main components highlighted in this paper: the pipeline, the Generator Service, the Deployment Service, and the Global ByteDance Resource Orchestration Service (gROS), as shown in Figure 2. One core concept in ByteRollout is **deployment**. A deployment consists of a list of resources that are scheduled for deployment in the intended request, along with additional information such as namespace and organization. To better support new data center bring-up scenarios that may require resources from multiple deployments, ByteRollout allows users to submit a batch of deployments in a single request.

BYTEROLLOUT is able to receive configuration files through an input request from three resources: Software Development Kit (SDK), Command Line Interface (CLI), and open APIs. The BYTEROLLOUT SDK and open APIs enable users to call the service directly from their code, while the CLI provides a user-friendly interface for developers to input commands and trigger the deployment process. Through an input service, users can trigger a BYTEROLLOUT pipeline consisting of a BYTEROLLOUT Register Atom, a BYTEROLLOUT Actuation Atom, and a BYTEROLLOUT Query Atom to initiate the deployment process. The BYTEROLLOUT Register Atom collects user input and sends the information to the Generator Service to create deployment and resource specifications. It also highlights the differences between the new and old specifications, allowing users to review and confirm the intended rollout. Once users approve the deployment, the BYTEROLLOUT Actuation Atom forwards the request to the Deployment Service, which manages database read/write operations, request throttling, and sequentially triggers each deployment. Next, the Global ROS processes the deployment by distributing resources based on their placement and forwarding them to the Regional ROS (rROS). The rROS is responsible for dependency resolution, scheduling, throttling, rollback, and the actual execution of user specifications. The BYTEROLLOUT Query Atom collects the status updates and display the ongoing tickets during this process. The components of BYTEROLLOUT generator and Deployment Services use an RPC framework named Kitex [54]. We will introduce the design of these components in a more detailed way in the following subsections.

A. *SoT File of* BYTEROLLOUT

The SoT (source of truth) file of BYTEROLLOUT to describe the resource deployment intent is designed in YAML format, while the fields are defined in Protobuf [52]. The benefits of using Protobuf include efficient serialization, scalability, and cross-platform compatibility, while also aligning with the coding practices of the teams.

Each resource is assigned an SoT file and consists of two main components: "metadata" and "spec". The metadata field defines the resource's logical ID, which must be globally unique within the deployment. It also includes additional information such as the resource type and the logical IDs of any dependent resources. The spec field consists of two main parts: "template" and "regionalResources". The template field contains global information about the resource that is shared across all regional resources. This may include details such as the host type, required CPU count, code repository version, and upgrade strategy. The regionalResources field specifies the configuration for each regional resource in the field of "propertyOverrides", along with its primary identifier to uniquely distinguish it.

An example of the SoT configuration file is shown in Listing 1. It defines a TCE Cluster resource with two regional clusters located in the SG1 and Maliva data centers. Its metadata field contains the resource's logical ID, type, and the logical IDs of its dependent resources. The resource has two regional resources that share common attributes such as host type, required GPU count, image version, and upgrade strategy. However, they also have region-specific attributes. For example, the SG1 cluster uses a more advanced code version, "1.0.0.48", compared to "1.0.0.47" in Maliva while the Maliva cluster requires a larger capacity, with a CPU count of 2 and a memory count of 3. When users need to modify a cluster, changes intended for a specific regional cluster should be made in the propertyOverrides section of the SoT file, such as updating the CPU count from "2" to "4". Conversely, changes that apply to all clusters should be made in the template section, such as renaming the cluster from "default" to "stress". Once users have their SoT files for the resources ready, approved, and merged into the codebase, they can submit an input request. This request should include the locations of these files in the repository, as well as deployment information, such as the deployment name.

Listing 1: SoT file example of a TCE Cluster resource

```
metadata:
  logicalId: demo.kitex.xianhao.tce.cluster
  type: TCE_CLUSTER
  dependencies:
    - demo.kitex.xianhao.tce.service
spec:
  regionalResources:
    VGeo-RoW|Singapore-Central|sg1:
      primaryIdentifier:
        Meta:
          Id: "4363735"
      propertyOverrides:
        Runtime:
          RepoInfo:
          - Name: toutiao/demo/xianhao_kitex
            ScmRepoId: "314438"
            Version: 1.0.0.48
    VGeo-RoW|US-East|maliva:
      primaryIdentifier:
        Meta:
          Id: "2706443"
      propertyOverrides:
        Resource:
          Cpu: 2
          Mem: 3
  template:
    Meta:
      HostType: docker
      Name: default
      Service: "1611598"
      ServiceEnv: prod
      ServicePriority: normal
      ServicePsm: demo.kitex.xianhao
    Resource:
      Cpu: 1
      Gpu: 0
      Mem: 2
      Package: ""
      ResourceType: custom
      Socket: 0
    Runtime:
      AmsTag: ""
      CanaryWeight: 10
```

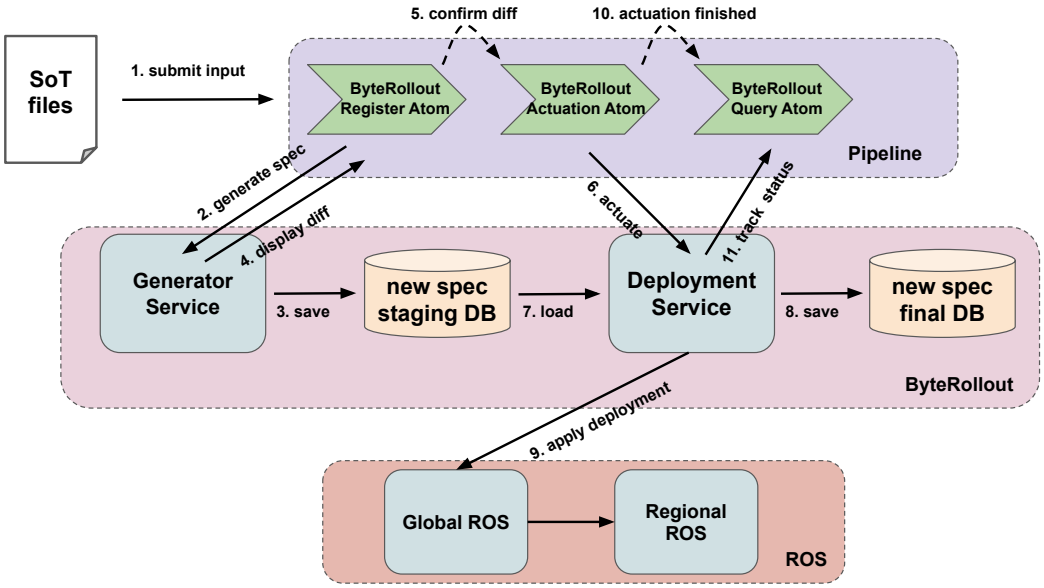

Fig. 2: Brief design of BYTEROLLOUT.

```
  HostuniqNum: 0
  ImageTag: tce
  ImageVersion: 1.0.0.74
  IsHostuniq: false
  IsStateful: false
  MaxFailureFraction: 0
  RepoInfo:
    - Name: toutiao/demo/xianhao_kitex
      ScmRepoId: "314438"
      Version: 1.0.0.47
  ShardNum: 0
  ShardParams: {}
  Weight: 10
Upgrade:
  CanarySurgePercent: 100
  IsStandaloneRelease: false
  MinReadySecond: 10
  StandaloneReason: ""
  SurgePercent: 25
  TerminationSecond: 30
  UpdateAction: restart
  UpdateMode: delete_first
  UpdateStrategy: RollingUpdate
```

B. BYTEROLLOUT *Pipeline Atoms*

BYTEROLLOUT utilizes ByteDance's pipelines as the user interface to display and track the status of the deployment workflow. The pipeline consists of a set of atoms, where each atom is a unit service that performs a group of related tasks. This structure enables the pipeline to integrate seamlessly with other platforms [22]. BYTEROLLOUT provides three atoms that are highlighted in this paper: Register Atom, Actuation Atom, and Query Atom. The BYTEROLLOUT Register Atom interacts with the Generator Service to create deployments based on the received inputs and displays the differences between the new deployment and the previous deployment with the same deployment name. The pipeline will be paused at the Register Atom phase until users approve the potential deployment. Once users confirm that the displayed differences meet expectations, the Actuation Atom is triggered, calling the Deployment Service to apply the deployment previously created by the Deployment Service. The actuation atom can also receive deployment sequence information as input, enabling the deployment of resources to follow a specified order based on resource types or regions. After the deployment is successfully initiated, the BYTEROLLOUT Query Atom continuously performs query operations until the deployment is marked as successful, failed, or canceled. Additionally, the atom displays ongoing platform tickets, allowing users to gain a more detailed understanding of the deployment progress.

C. BYTEROLLOUT *Services*

In this paper, we focus on two key services in BYTEROLLOUT: the Generator Service and the Deployment Service. Additionally, BYTEROLLOUT includes other services, such as the Input Service, which triggers the pipeline, and the Entry Service, which manages HTTP calls. The database we select for BYTEROLLOUT is an internally developed database based on MongoDB [5]. MongoDB is chosen as the preferred database solution due to its inherent flexibility and efficient querying capabilities. In particular, the absence of rigid relationships among files enables each file to be treated as an independent record within the database.

The BYTEROLLOUT Generator Service is responsible for parsing the SoT file contents from the specified repository and path, translating each YAML file into its corresponding resource, and assembling the resources into a deployment.

The Generator Service offers three key features. First, in the case of a new data center bring-up, numerous resources are involved, each with its own SoT file, *e.g.,* a typical new data center bring-up can involve thousands of resources and the number of SoT files can be ten times the number of resources. As a result, the Generator Service must read a large number of files from the codebase, which can create a bottleneck for the version control system. This process can slow down the overall deployment, and since SoT files often require frequent minor updates, the issue can become even more pronounced. To mitigate this issue, the Generator Service caches the required SoT files for the data center bring-up process and applies incremental updates from the version control system only when necessary. This reduces the load on the version control system and improves efficiency. Next, certain operations are repeated across all SoT files during a data center bring-up. For example, when adding a new data center, every SoT file must include a new key in the "regionalResources" field to reference the new data center. Similarly, once all TCE clusters are ready, SRE teams may need to increase capacity, requiring simultaneous updates to the corresponding CPU and memory counts across all relevant files. Therefore, the Generator Service provides a "batch-edit" operation, enabling users to apply the same change across multiple specified SoT files with a single request, streamlining the update process.

Moreover, SoT configuration files can be complex and contain a significant amount of information that may be irrelevant to certain users. For example, users from the capacity management team may only be concerned with CPU count and memory count, while users from the daily development team may primarily focus on the code version. As a result, there is a need to simplify the SoT file and allow for customized configuration views tailored to specific user needs. To address this issue, the Generator Service supports customized generators as plugins alongside the default generators. These customized generators, which can be developed by other customer teams, allow users to translate their own customized SoT files into the standard resource format defined by Protobuf, efficiently improving the readability for users. After successfully generating the deployment with all resources and making it ready for review, the Generator Service stores it in a staging database.

After the user confirms the potential deployment, the ByteRollout Deployment Service receives the actuation request and loads the deployment from the staging database. It then applies the deployment to gROS and tracks its status. Additionally, the service interacts with the final database to store the deployment and its related historical information. Furthermore, it enables users to cancel or roll back an ongoing deployment through gROS.

V. Evaluation

In this section, we discuss how ByteRollout was utilized in the bring-up of three industrial new data centers and analyze the cost savings achieved through the procedures. We did not apply a similar IaC technique in these data center bring-ups due to cost concerns. As a result, we can only compare ByteRollout with the tools that were used in previous data center bring-ups.

A. Evaluation Process

From January 2024 to December 2024, ByteRollout was successfully used in the bring-up of three new data centers. However, due to company policies, some statistics related to the process cannot be disclosed in this paper, and the real-world locations of these data centers cannot be referenced. We summarized the details of each data center bring-up in Table I. To measure the effectiveness of ByteRollout, we calculated the difference in developer days spent on bringing up a data center using ByteRollout compared to the previous experience of bringing up a similar-sized data center by the same team. With the saved developer days and reduced machine waiting time, we can also estimate the total cost savings in dollars, providing a clearer picture of ByteRollout's efficiency and impact.

The first data center bring-up using ByteRollout was carried out by the global payment team. The primary resource involved in this evaluation was the Relational Database Service (RDS). As part of this evaluation, we also conducted a deployment rehearsal involving approximately 20 TCE services with their respective clusters. The second ByteRollout evaluation was for Product B, where it was used to set up a new data center in Europe. ByteRollout supported the deployment of over 4000 TCE Cluster resources in this process. After the initial deployment, the team used ByteRollout to increase the capacity of around 3000 TCE Clusters. Additionally, over 4000 traffic governance Neptune resources were migrated through our system. The third data center bring-up evaluation was operated by Product C team. A number of different categories of resources were supported by ByteRollout during the process, including TCE, RDS, Redis [9], and FaaS [51]. Over 500 RDS databases, 70000 RDS tables, and 3900 TCE Clusters were successfully brought up though ByteRollout.

B. Results

We first compared the bring-up progress of data center A, which adopted ByteRollout, with a similarly sized data center A*, which did not. The results showed that ByteRollout significantly reduced the developer days involved in the process from over 40 to less than 1.5. Additionally, the deployment time span was reduced from over 2 months to just 12 business days. We further evaluated how ByteRollout contributed to these cost savings and identified two key factors: (1) automated deployment with SoT Files – ByteRollout replaced the previous manual triggering and categorization steps with automated deployment using SoT files, significantly reducing human effort, (2) automated ticket generation – ByteRollout streamlined the process by automatically generating deployment review tickets, eliminating the need for manual approval and expediting workflow execution.

TABLE I: Data center bring-up statistics.

Data Center	Team	Resources	Count	Expand Capacity?
A	Product A	RDS, TCE	200 RDS databases, 9000+ RDS tables 20 TCE Services	No
B	Product B	TCE, Neptune	4000+ TCE Clusters 4400+ Neptune resources	Yes
C	Product C	RDS, TCE, Redis, FaaS	500+ RDS databases, 70000+ RDS tables ∼4000 TCE Clusters 500+ Redis 1200 FaaS	Yes

From the evaluation of data center B, operated by Product B, we found that BYTEROLLOUT significantly accelerated the bring-up process, reducing the overall timespan from 6 months to 3 months. Specifically, BYTEROLLOUT shortened the deployment phase from 2.5 months to 1 month, saving the company tens of millions of dollars. The TCE deployment period was reduced from 4 weeks to 3 weeks, with the success rate increasing from 60% to 73% (with the remaining failures unrelated to infrastructure). In total, BYTEROLLOUT saved up to 312 developer days for business teams during deployment and approximately 30 SRE team days in the TCE scale-up phase, achieving an accuracy rate of over 95%. Additionally, it took only 53 minutes to scale up 3,000 clusters, around 90 minutes to deploy 4,000 TCE clusters, and less than 2 hours to actuate over 4000 Neptune resources. We analyzed the key factors behind BYTEROLLOUT 's cost-saving effectiveness and identified two primary reasons. First, its high deployment success rate significantly reduces the time required for examination and debugging. Second, the automatic retry feature minimizes manual intervention and accelerates the overall deployment process, further reducing operational duration.

In the final evaluation of the data center C bring-up, BYTEROLLOUT successfully saved over 572 developer days in total. Among the various resources involved, the deployment of RDS tables contributed the most significant savings, reducing over 400 developer days, while the deployment of TCE clusters ranked second, saving approximately 90 developer days. This further demonstrates that BYTEROLLOUT can lower costs during data center deployments by automating the process and minimizing the failure rate. Considering that BYTEROLLOUT will support more resources in the future and experience fewer bugs as the system matures, it will continue to contribute to cost savings for data center bring-ups. This is due to the reusable nature of Infrastructure as Code.

VI. DISCUSSION

In this section, we discuss the challenges encountered and lessons learned during the process of building BYTEROLLOUT and achieving its standards to meet user requirements. We also discuss the forthcoming challenges that need to be addressed in the system and evolution direction in the future.

A. Lesson Learned

The first lesson we learned was how to address the requirement for handling a high concurrency request rate within a short period during the data center bring-up process. Therefore, BYTEROLLOUT enhances its's scalability in two key ways. The initial approach involves storing the SoT configuration files in cache and performing only incremental updates through the version control system after a deployment request is submitted. This solution significantly reduces the time required to read the SoT file contents from the remote codebase, thereby shortening the overall deployment generation duration. The second approach involves replacing the synchronous mode with an asynchronous one, allowing users to proceed without waiting for a response while the deployment is still in progress. This reduces the likelihood of timeout failures, especially considering the already lengthy call chain. To enhance user support in monitoring deployment status, BYTEROLLOUT implements a status tracker and provides a query webpage for users to access real-time updates.

The second lesson we learned during the implementation of BYTEROLLOUT is the flexibility and readability of the SoT file as the system input. The SoT files are typically long and complex, as they contain all the necessary information to define a resource. This makes it challenging for a single user to fully comprehend all of the contents. Additionally, the centralized team may only be concerned with specific attributes of a resource, such as capacity or placement. As a result, users may require a customized version of the SoT file to better manage resources within their area of expertise. This necessitates BYTEROLLOUT to support multiple input formats. To address this, BYTEROLLOUT allows customized generators as plugins to its Generator Service, enabling the translation of customized SoT file formats into a predefined standard resource description format.

Another lesson we learned during the design of BYTEROLLOUT is how to overcome the security issue. BYTEROLLOUT faces both the issues of authorization and authentication: the system must identify the input user and also decide what permission the user should have. Besides, an individual user may lack the necessary permissions to deploy all resources during the data center bring-up process. To address this, BYTEROLLOUT accepts both the user's JSON Web Token and the service account token, allowing teams to grant all necessary permissions to their service account. Other lessons we learned include the input diversity requirement, the trade-off between Protobuf and JSON, and the appropriate selection of databases.

B. Forthcoming Challenges and Evolution Direction

One challenge that needs to be addressed is how to detect errors and provide useful suggestions when editing SoT files, as it is difficult to identify errors while modifying

the configuration file. Currently, the integrated development environment (IDE) does not support error detection when editing BYTEROLLOUT SoT files. Therefore, future plugins could be developed to detect typos and incorrect inputs within the configuration files. This also introduces another challenge: locating bugs in the configuration file and providing automatic fix suggestions. This issue could potentially be resolved by integrating artificial intelligence tools to assist in the editing and debugging process. Given the high reusability of SoT configurations, one potential direction for evolution is to extend the use of BYTEROLLOUT to daily resource deployments (Continuous Deployment), rather than limiting it solely to new data center bring-ups. One major benefit is that all changes will be reflected in the SoT files within the codebase, allowing for code reviews from a security perspective and making it easier to implement version control. However, this approach can also introduce some conflicts. One major issue is the consistency of the SoT file. Since there are multiple entry points for making changes to deployments, such as the resource provider platform or the pipeline atom, the SoT file may not always be up-to-date. As a result, when there is a drift, applying the deployment through the SoT files could lead to consistency issues. To address this issue, we need to consolidate the entry points and ensure that all resource changes are reflected in the SoT files before they can take effect.

Additionally, one benefit of converting infrastructure configuration into code is that it becomes easier for AI agents utilizing large language model (LLM) technology to parse and understand the information. Therefore, adopting infrastructure as code facilitates the potential for AI agents to manage all resources efficiently. This enables users, such as software developers and SREs, to interact directly with the AI agent by communicating prompts about the desired infrastructure changes, allowing the AI agent to handle the complex configuration tasks. This also enables developer efficiency teams to build AI agents that monitor metrics of the resources and make adjustments accordingly. Apart from that, translating the configuration file into code enables users to edit it alongside the business code within an integrated development environment. This approach allows users to manage both infrastructure and business code in a synchronized and unified manner. Other challenges include the situation where a single resource is applied simultaneously in different deployments, which necessitates locking the resource during an ongoing deployment to prevent conflicts. To apply BYTEROLLOUT in the daily software engineering deployment cycle, we also need to develop a more comprehensive user interface that allows users to register deployments by specifying resource identifiers. Additionally, the system should integrate the code review and the status query process. User feedback also reflects a feature request to provide options to fast rollback.

VII. THREATS TO VALIDITY

A. Internal Validity

To guard internal validity, we reported the evaluation of BYTEROLLOUT in three different data center bring-ups and all the three evaluations include a large amount of resources in different categories. However, we are unable to compare the effectiveness of using BYTEROLLOUT versus not using it in a single data center bring-up, as doing so would result in significant waste. Therefore, we compare the evaluation results with the same team's previous experience in data center bring-ups to minimize the impact of this comparison. Also, we collected the data from three events of new data center bring-ups to avoid contingency. Our observed results may have been influenced by the load experienced in the build server at the time. However, we consider this potential impact to be minimal, and it may apply equally to experiences with or without BYTEROLLOUT. Another factor that may influence the findings is the logical inferences and conclusions drawn from the research results. Besides, although we strived to minimize differences between the compared processes aside from the adoption of BYTEROLLOUT, certain variables (*e.g.,* build server load) could not be fully controlled.

B. External Validity

To increase external validity, we selected all resources that were involved in the data center bring-up supported by BYTEROLLOUT. The resources include different types of the software engineering technologies of services, clusters, database and cache with different programming languages. Our observations may slightly vary for separate software projects, but our goal was to derive general observations for a real-world population of software projects. Given that the data center bring-up events we collected were all from industrial areas, the results may not be always applicable to some smaller open-source projects.

C. Construct Validity

A threat to the validity of the construct is whether we studied software projects that are similar to those in the real world. However, we have selected the maximum diversity of resources across different categories and service sizes to ensure a comprehensive evaluation. Additionally, the results may be influenced by differences in team cultures and collaboration patterns. However, given the scale of the data centers and the fact that the evaluated data center bring-ups involve teams from different regions and time zones, the benefits of adopting Infrastructure as Code remain substantial and well-supported.

VIII. RELATED WORK

A. Infrastructure as Code

Infrastructure as Code (IaC) has been popular as automation technologies [36]. Four topics studied in IaC-related publications are identified by existing work [41]: (i) framework/tool for Infrastructure as Code; (ii) use of Infrastructure as Code; (iii) empirical study related to Infrastructure as Code; and (iv) testing in Infrastructure as Code. Guerriero *et al.* explore the adoption, support and challenges of IaC and highlight the need for more research in the field: The support provided by currently available tools is still limited, and developers feel the need for novel techniques to test and maintain IaC code

[19]. Other existing work focuses on helping practitioners improve the quality of Infrastructure as Code (IaC) scripts by identifying development activities related to defective IaC scripts [40] and avoiding insecure coding practices while developing IaC scripts through an empirical study of security smells in IaC scripts [42].

Rong *et al.* proposed an approach named Open Infrastructure as Code (OpenIaC), which is an attempt to provide a common open forum to integrate and build on advances in cloud computing and blockchain to address the needs of modern information architectures [44]. Also, existing work finds that IaC becomes more popular in Stack Overflow and the related topics include server configuration, policy configuration, networking, deployment pipelines, variable management, templating, and file management [6]. Other research focuses on assisting with the editing of configuration files in the field of Infrastructure as Code (IaC), aiming to improve accuracy, reduce errors, and enhance usability [28]. Palma *et al.* aimed at assessing the role of product and process metrics when predicting defective IaC scripts [13] while other work also working on ensuring the quality of IaC [7], [30], [38], [39].

Our BYTEROLLOUT platform is designed based on the unique context of developing environment at ByteDance and it also has many differences between what is Terraform, *e.g.,* we allow a pipeline engine to drive our deployment workflow and accepts customized generators as plugins to contribute to our Generator Service. We also propose the first evaluation on how Infrastructure as Code can accelerate the data center bring-up events and we compare the effects in three real-world data center bring-up practices.

B. Approaches to Improve Continuous Deployment

Researchers worked to explain why Continuous Delivery (CDE) and Continuous Deployment (CD) are adopted and report the huge benefits and challenges involved [10]. Chen *et al.* presented strategies to help overcome the challenges of CDE [11] and Virmani *et al.* also explained how to bridge the gap of CDE to promote the adoption [53]. Savor *et al.* explained how CD has been achieved at Facebook [48]. Existing work [12] also examined the challenges faced by organizations when adopting CD, as well as strategies to mitigate these challenges. Yang *et al.* explored two types of workflow in CD including Docker Hub auto-builds Workflows and CI-based Workflows [58]. Other works also explained the difference between CD and CDE and their own approaches and challenges [49], [50].

As a related effort to improve Continuous Deployment, Gallaba *et al.* explore ways to improve the robustness and efficiency of CD processes [17]. Also, existing work aims to aid product teams in improving their deployment process through characterizing experimentation in CD [27]. Researchers also present a machine learning-based framework that systematically improves pipeline performance through predictive modeling to optimize the CD pipelines [14]. Rahman and Williams took advantage of two text mining techniques to extract text features from IaC scripts to characterize defective IaC scripts in Continuous Deployment [43].

Another related practice is Continuous Integration (CI), which typically precedes Continuous Deployment (CD). CI often includes steps such as code compilation, unit testing, integration testing, and testing in an offline environment. To accelerate CI, existing work aim at speeding up its feedback by prioritizing its tasks [15], [16], [31], [33], [37], [47], [60], performing test selection [16], [18], [20], [32], [45], [46], [55], [56], [57], [59], and reducing the total number of builds that get executed [1], [2], [23], [25], [26], [24].

Our work adopts the concept of Infrastructure as Code, which is widely used in Continuous Deployment. We applied this approach to data center bring-ups and implemented BYTEROLLOUT. The evaluation results show that Infrastructure as Code can save hundreds of developer day and months of machine idle time when building a new data center and thus save millions of dollars for the company. In future work, we plan to expand BYTEROLLOUT as a platform to support daily resource deployments as part of Continuous Deployment and apply Large Language Model tools to edit and debug the IaC configuration scripts.

IX. CONCLUSIONS AND FUTURE WORK

In this article, we introduced the process of bringing up a new data center with various categories of resources in a large tech company like ByteDance. We also included the challenges and existing problems that can cause human effort waste throughout the process. We also introduced our design of BYTEROLLOUT, a platform to deploy resources to a new data center using the concept of Infrastructure as Code. We evaluated BYTEROLLOUT with three data center bring-up events to understand how Infrastructure as Code can accelerate the entire process. We compared the time span and developer days cost for data center bring-ups that adopted BYTEROLLOUT against those that did not, based on three separate evaluation cases. We found that Infrastructure as Code can accelerate the data center bring-up process significantly. Our findings can shed light on the design of future tools to address the challenges. Finally, we discuss what we had learned during the design of BYTEROLLOUT and the gap between the initial design and how developers would like to use the tool. We also provided a set of challenges or pain points that may require future work to address and emphasized the evolution direction to use BYTEROLLOUT as a Continuous Deployment platform. We lay out plans to simplify the edit and debug process of the configuration files to enhance their flexibility and practicality using LLM models. In the future, we will work on using BYTEROLLOUT to complement the current existing Continuous Deployment platform to take advantage of the benefits of code review and version control. We will also work on a more comprehensive webpage to serve as the user interface for better user experience.

REFERENCES

[1] R. Abdalkareem, S. Mujahid, and E. Shihab. A machine learning approach to improve the detection of ci skip commits. *IEEE Transactions on Software Engineering (TSE)*, 2020.

[2] R. Abdalkareem, S. Mujahid, E. Shihab, and J. Rilling. Which commits can be ci skipped? *IEEE Transactions on Software Engineering*, 2019.

[3] M. Arif, M. M. Rafique, S.-H. Lim, and Z. Malik. Infrastructure-aware tensorflow for heterogeneous datacenters. In *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 1–8. IEEE, 2020.

[4] W.-H. Bai, J.-Q. Xi, J.-X. Zhu, and S.-W. Huang. Performance analysis of heterogeneous data centers in cloud computing using a complex queuing model. *Mathematical Problems in Engineering*, 2015(1):980945, 2015.

[5] K. Banker, D. Garrett, P. Bakkum, and S. Verch. *MongoDB in action: covers MongoDB version 3.0*. Simon and Schuster, 2016.

[6] M. Begoug, N. Bessghaier, A. Ouni, E. A. AlOmar, and M. W. Mkaouer. What do infrastructure-as-code practitioners discuss: An empirical study on stack overflow. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12. IEEE, 2023.

[7] M. Begoug, M. Chouchen, and A. Ouni. Terrametrics: an open source tool for infrastructure-as-code (iac) quality metrics in terraform. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, pages 450–454, 2024.

[8] Y. Brikman. *Terraform: up and running: writing infrastructure as code*. " O'Reilly Media, Inc.", 2022.

[9] J. Carlson. *Redis in action*. Simon and Schuster, 2013.

[10] L. Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE software*, 32(2):50–54, 2015.

[11] L. Chen. Continuous delivery: overcoming adoption challenges. *Journal of Systems and Software*, 128:72–86, 2017.

[12] G. G. Claps, R. B. Svensson, and A. Aurum. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology*, 57:21–31, 2015.

[13] S. Dalla Palma, D. Di Nucci, F. Palomba, and D. A. Tamburri. Within-project defect prediction of infrastructure-as-code using product and process metrics. *IEEE Transactions on Software Engineering*, 48(6):2086–2104, 2021.

[14] S. Dileepkumar and J. Mathew. Optimizing continuous integration and continuous deployment pipelines with machine learning: Enhancing performance and predicting failures. *Advances in Science and Technology Research Journal*, 19(3):108–120, 2025.

[15] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, 28(2):159–182, 2002.

[16] S. Elbaum, G. Rothermel, and J. Penix. Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 235–245, 2014.

[17] K. Gallaba. Improving the robustness and efficiency of continuous integration and deployment. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 619–623. IEEE, 2019.

[18] M. Gligoric, L. Eloussi, and D. Marinov. Practical regression test selection with dynamic file dependencies. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 211–222, 2015.

[19] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba. Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *2019 IEEE international conference on software maintenance and evolution (ICSME)*, pages 580–589. IEEE, 2019.

[20] K. Herzig, M. Greiler, J. Czerwonka, and B. Murphy. The art of testing less without sacrificing quality. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 483–493. IEEE, 2015.

[21] J. Hwang, S. Zeng, F. y Wu, and T. Wood. Benefits and challenges of managing heterogeneous data centers. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 1060–1065. IEEE, 2013.

[22] X. Jin, Y. Feng, C. Wang, Y. Liu, Y. Hu, Y. Gao, K. Xia, and L. Guo. Pipelineascode: A ci/cd workflow management system through configuration files at bytedance. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1011–1022. IEEE, 2024.

[23] X. Jin and F. Servant. A cost-efficient approach to building in continuous integration. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 13–25. IEEE, 2020.

[24] X. Jin and F. Servant. What helped, and what did not? An Evaluation of the Strategies to Improve Continuous Integration, Mar. 2020. Available at https://doi.org/10.5281/zenodo.4372963.

[25] X. Jin and F. Servant. Which builds are really safe to skip? maximizing failure observation for build selection in continuous integration. *Journal of Systems and Software*, 188:111292, 2022.

[26] X. Jin and F. Servant. Hybridcisave: A combined build and test selection approach in continuous integration. *ACM Transactions on Software Engineering and Methodology*, 32(4):1–39, 2023.

[27] K. Kevic, B. Murphy, L. Williams, and J. Beckmann. Characterizing experimentation in continuous deployment: a case study on bing. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 123–132. IEEE, 2017.

[28] P. T. Kon, J. Liu, Y. Qiu, W. Fan, T. He, L. Lin, H. Zhang, O. M. Park, G. S. Elengikal, Y. Kang, et al. Iac-eval: A code generation benchmark for cloud infrastructure-as-code programs. *Advances in Neural Information Processing Systems*, 37:134488–134506, 2024.

[29] M. Leppänen, S. Mäkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistö. The highways and country roads to continuous deployment. *Ieee software*, 32(2):64–72, 2015.

[30] E. Low, C. Cheh, and B. Chen. Repairing infrastructure-as-code using large language models. In *2024 IEEE Secure Development Conference (SecDev)*, pages 20–27. IEEE, 2024.

[31] Q. Luo, K. Moran, D. Poshyvanyk, and M. Di Penta. Assessing test case prioritization on real faults and mutants. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 240–251. IEEE, 2018.

[32] M. Machalica, A. Samylkin, M. Porth, and S. Chandra. Predictive test selection. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 91–100. IEEE, 2019.

[33] D. Marijan, A. Gotlieb, and S. Sen. Test case prioritization for continuous regression testing: An industrial case study. In *2013 IEEE International Conference on Software Maintenance*, pages 540–543. IEEE, 2013.

[34] P. Mell, T. Grance, et al. The nist definition of cloud computing. 2011.

[35] K. Morris. *Infrastructure as code: managing servers in the cloud.* " O'Reilly Media, Inc.", 2016.

[36] K. Morris. *Infrastructure as code*. O'Reilly Media, 2020.

[37] S. Mostafa, X. Wang, and T. Xie. Perfranker: prioritization of performance regression tests for collection-intensive software. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 23–34, 2017.

[38] E. Ntentos, N. E. Lueger, G. Simhandl, U. Zdun, S. Schneider, R. Scandariato, and N. E. Díaz Ferreyra. On the understandability of design-level security practices in infrastructure-as-code scripts and deployment architectures. *ACM Transactions on Software Engineering and Methodology*, 34(1):1–37, 2025.

[39] R. Opdebeeck, A. Zerouali, and C. De Roover. Control and data flow in security smell detection for infrastructure as code: Is it worth the effort? In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 534–545. IEEE, 2023.

[40] A. Rahman, E. Farhana, and L. Williams. The 'as code'activities: development anti-patterns for infrastructure as code. *Empirical Software Engineering*, 25:3430–3467, 2020.

[41] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams. A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65–77, 2019.

[42] A. Rahman, C. Parnin, and L. Williams. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 164–175. IEEE, 2019.

[43] A. Rahman and L. Williams. Characterizing defective configuration scripts used for continuous deployment. In *2018 IEEE 11th International conference on software testing, verification and validation (ICST)*, pages 34–45. IEEE, 2018.

[44] C. Rong, J. Geng, T. J. Hacker, H. Bryhni, and M. G. Jaatun. Openiac: open infrastructure as code-the network is my computer. *Journal of Cloud Computing*, 11(1):12, 2022.

[45] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on software engineering*, 22(8):529–551, 1996.

[46] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(2):173–210, 1997.

[47] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, 27(10):929–948, 2001.

[48] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm. Continuous deployment at facebook and oanda. In *Proceedings of the 38th International Conference on software engineering companion*, pages 21–30, 2016.

[49] M. Shahin, M. A. Babar, M. Zahedi, and L. Zhu. Beyond continuous delivery: an empirical investigation of continuous deployment challenges. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 111–120. IEEE, 2017.

[50] M. Shahin, M. A. Babar, and L. Zhu. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE access*, 5:3909–3943, 2017.

[51] M. Shahrad, J. Balkind, and D. Wentzlaff. Architectural implications of function-as-a-service computing. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, pages 1063–1075, 2019.

[52] K. Varda. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Available at least as early as Jul*, 72:23, 2008.

[53] M. Virmani. Understanding devops & bridging the gap from continuous integration to continuous delivery. In *Fifth international conference on the innovative computing technology (intech 2015)*, pages 78–82. IEEE, 2015.

[54] Y. Wen, G. Cheng, S. Deng, and J. Yin. Characterizing and synthesizing the workflow structure of microservices in bytedance cloud. *Journal of Software: Evolution and Process*, 34(8):e2467, 2022.

[55] S. Yoo and M. Harman. Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 international symposium on Software testing and analysis*, pages 140–150. ACM, 2007.

[56] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

[57] L. Zhang. Hybrid regression test selection. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 199–209. IEEE, 2018.

[58] Y. Zhang, B. Vasilescu, H. Wang, and V. Filkov. One size does not fit all: an empirical study of containerized continuous deployment workflows. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 295–306, 2018.

[59] C. Zhu, O. Legunsen, A. Shi, and M. Gligoric. A framework for checking regression test selection tools. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 430–441. IEEE, 2019.

[60] Y. Zhu, E. Shihab, and P. C. Rigby. Test re-prioritization in continuous testing environments. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 69–79. IEEE, 2018.