Challenges in Estimating Deployment Timelines for Offline Infrastructure-as-Code Infrastructure Deployment.

**Overview**

The team is in the process of deploying a complex infrastructure consisting of a mix of technologies that have had little to no use in the business currently. The infrastructure consists of Fedora based, development machines that act as self-contained Kubernetes clusters using K3s, and a full Kubernetes cluster that, in previous deployments has proven difficult on an air-gapped VMware vCloud environment. It will be using a community based automated deployment that will install all required components in a single deployment. This will require a high amount of troubleshooting due to the dependency on internet access. This system integrates Terraform, Ansible, and Kubespray, technologies that we are actively learning and implementing. Additionally, an offline GitLab instance will serve as the CI/CD backbone for both the development VMs and the Kubernetes cluster.

Key Components and Their Challenges in an Offline Environment

1. Terraform (Infrastructure as Code for VM Provisioning)
   * Terraform is being used to define and automate the provisioning of virtual machines.
   * In an air-gapped environment, Terraform modules and provider plugins need to be manually sourced, validated, and imported, increasing setup time.
   * Debugging errors is more difficult without direct internet access for troubleshooting or retrieving updated modules.
2. Ansible (Configuration Management and Automation)
   * Ansible is responsible for configuring VMs and deploying necessary software packages.
   * Without internet access, role dependencies and collections must be pre-downloaded and manually synchronized.
   * Updating configurations requires iterative testing since patches and fixes cannot be fetched dynamically.
3. K3s (Lightweight Kubernetes for Development Machines)
   * K3s provides a lightweight Kubernetes distribution for our Fedora-based development environments.
   * All container images must be mirrored manually, which is a time-consuming process requiring careful dependency management.
   * Network configuration and DNS resolution in an air-gapped setup add extra complexity.
4. Kubespray (Kubernetes Cluster Deployment)
   * Kubespray automates the deployment of a full Kubernetes cluster across multiple VMs.
   * It relies on numerous Ansible playbooks and external dependencies, all of which must be mirrored and tested offline.
   * Troubleshooting errors in an air-gapped environment requires deep debugging without access to online resources.
5. GitLab (Offline CI/CD for Both Development and Kubernetes Infrastructure)
   * GitLab will serve as the foundation for version control and CI/CD automation.
   * Maintaining an offline GitLab instance requires regularly mirroring updates and security patches manually.
   * Pipeline dependencies must be sourced and stored internally, adding maintenance overhead.
6. VMware vCloud (Virtualization Platform for Hosting the Infrastructure)
   * VMware vCloud is the foundation of the infrastructure, providing virtualized compute resources.
   * Any integration with external APIs or plugins needs manual configuration.
   * Network isolation creates additional hurdles in ensuring communication between VMs and Kubernetes nodes.

Complications in Working Offline

* Dependency Management: All required packages, modules, and container images must be identified, downloaded externally, and transported into the air-gapped network before use.
* Limited Troubleshooting Resources: Without direct internet access, error resolution depends entirely on internal documentation and trial-and-error debugging.
* Update Constraints: Any required updates or patches need to be manually acquired and tested, delaying overall progress.
* Network Configuration Complexity: Ensuring connectivity between components in a fully isolated environment requires additional setup, such as configuring private DNS, proxies, and internal repositories.

**Conclusion**

Given the combination of new technology adoption, air-gapped constraints, infrastructure interdependencies, and the iterative nature of development, accurately estimating deployment timelines is inherently difficult. While progress is steady, unforeseen technical challenges and the need for repeated testing cycles make precise predictions unreliable. However, as we continue refining our processes, future deployments will become more predictable and efficient.