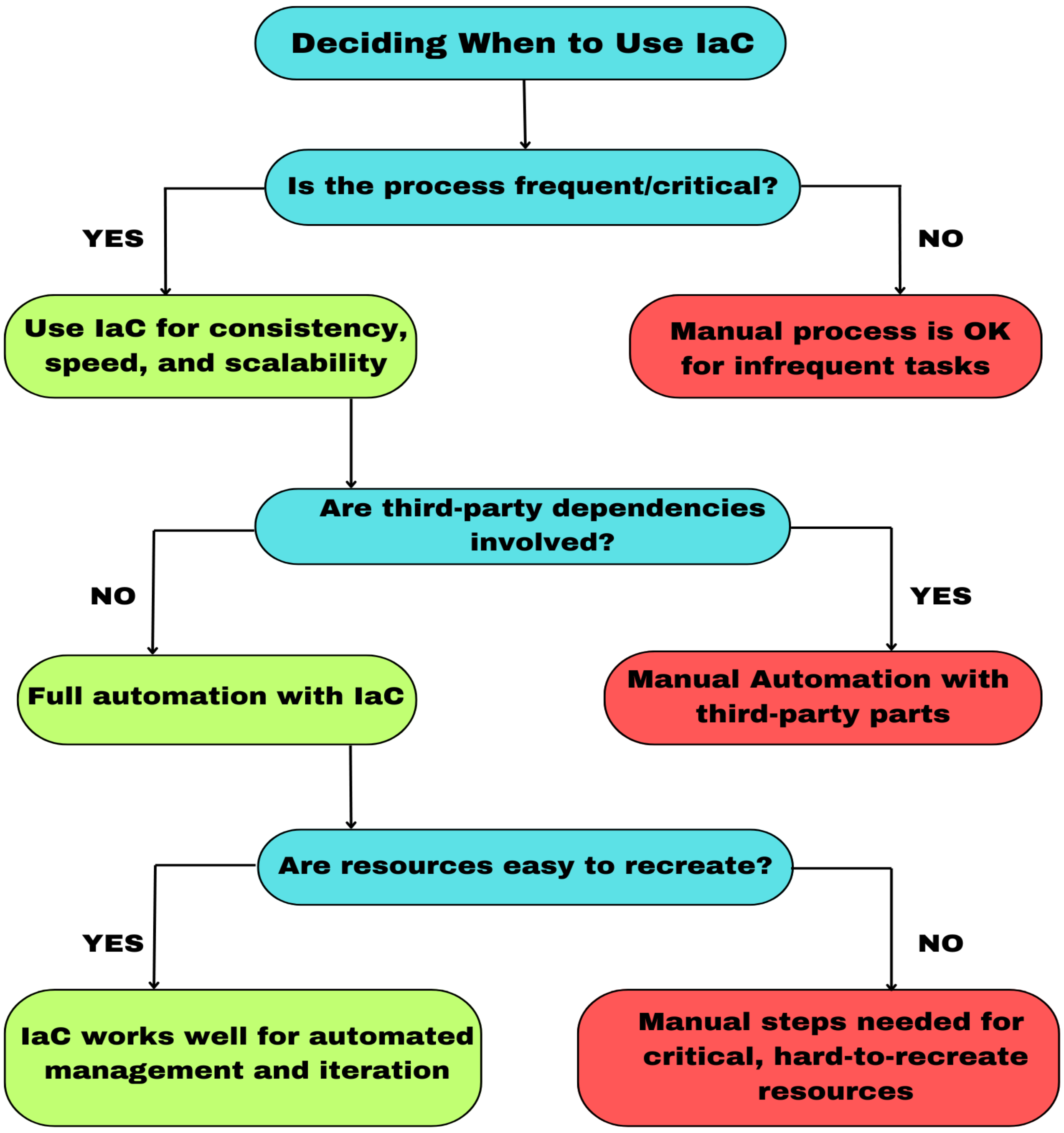
DevOps Shack

When and When Not to Use Infrastructure as Code


Deciding When to Use IaC
Is the process frequent/critical?
YES
NO
Use IaC for consistency, speed, and scalability
Manual process is OK for infrequent tasks
Are third-party dependencies involved?
NO
YES
Full automation with IaC
Manual Automation with third-party parts
Are resources easy to recreate?
YES
NO
IaC works well for automated management and iteration
Manual steps needed for critical, hard-to-recreate resources

Introduction

Striving for "100% Infrastructure as Code" has become a common goal for many DevOps teams today. Defining and managing nearly all infrastructure provisioning and configuration aspects through code is robust. It saves time, reduces manual toil, improves consistency, and minimizes human errors.

However, Infrastructure as Code (IaC) is not a panacea for every DevOps process or workflow. As Derek Ashmore, Application Transformation Principal at Asperitas, argues, aiming for 99% IaC coverage is often better than striving for 100%. Knowing when not to use IaC is just as important as leveraging it extensively.

The following explores the limitations of IaC and discusses how to identify processes that may be better handled manually, based on the Q&A session I had with Derek.

The Limitations of Infrastructure as Code

While IaC tooling has matured to the point where it's technically possible to automate virtually any infrastructure process, you can still do so. There are scenarios where an IaC approach can create more problems than it solves:

1. Infrequently performed processes (once or twice a year at most)
2. Workflows that rely on third-party resources outside your control
3. Operations involving resources that can't be easily recreated or relaunched

In these situations, the upfront and ongoing costs of developing and maintaining IaC can outweigh the time savings and other benefits it provides. Let's examine each of these in more detail.

Evaluating Infrequent Processes

For processes only performed once or twice per year, the value of automating them with IaC diminishes. Ashmore uses a simple formula to evaluate the cost-benefit:

1. Assume a 2-year "shelf life" for IaC before it needs significant maintenance.

2. Estimate 30% of initial development time/cost for that maintenance work.
3. Calculate the break-even point when the labor cost of IaC is less than that of manual execution.

"For infrequently executed manual processes, the break-even point is never reached," he explains. The labor required to develop, test, and maintain the IaC is at most the toil of executing the process manually, given how rarely it occurs.

An SSL certificate renewal that happens every year or two is an example. It's not hard to write code to automate the renewal, but it's also not time-consuming to do manually. In the interim, the certificate authority may change its process, breaking your IaC when you rerun it.

Working With Third-Party Dependencies

IaC is also problematic for workflows involving resources managed by third parties, like an interconnection between a public cloud and a colocation facility. "You can't fully manage operations you don't control," says Ashmore.

In this example, you rely on the colocation provider to establish the physical connection on their schedule. With control over the resource, automating its provisioning end-to-end with IaC is easier.

However, Ashmore still advises automating the portions of these workflows you control. "The same principles for evaluating the cost-effectiveness of that IaC apply," he notes. If the time savings of partial automation justify the development and maintenance costs, it may be worth pursuing.

Avoiding Resources That Can't Be Easily Recreated

Finally, be cautious about using IaC for resources that can only be recreated as part of an automated workflow. Secrets like passwords and encryption keys are a prime example.

While secret management tools often have features for recovering deleted secrets within a time window, they can't typically be undeleted if too much time has passed. This makes re-running your IaC workflows to manage those secrets challenging, which prevents effective iteration and testing.

Strategies for Maximizing IaC Value

So, how can DevOps teams get the most value from IaC while avoiding these pitfalls? Ashmore offers a few key recommendations:

- Focus on minimizing manual labor over time, not reaching 100% IaC. "The difference is subtle but significant."
- Implement automated testing to identify when changes to technologies, APIs, or policies have "broken" your IaC. This allows you to fix issues proactively before they impact the business.
- Design your IaC templates to be modular and easily maintained. While this will not prevent all maintenance, it will make it less costly.
- Continually reevaluate your IaC decisions as tools and platforms evolve. A process that doesn't make sense for IaC today may make sense in the future.

Conclusion

Infrastructure as Code is a game-changer for DevOps when applied smartly. By being selective about where and how you leverage IaC, you can get maximum value from it while avoiding unnecessarily high development and maintenance costs.

Remember, the goal is not to reach 100% IaC coverage. It's to minimize manual effort and maximize efficiency overall. Knowing when not to use IaC is a crucial part of achieving that objective.