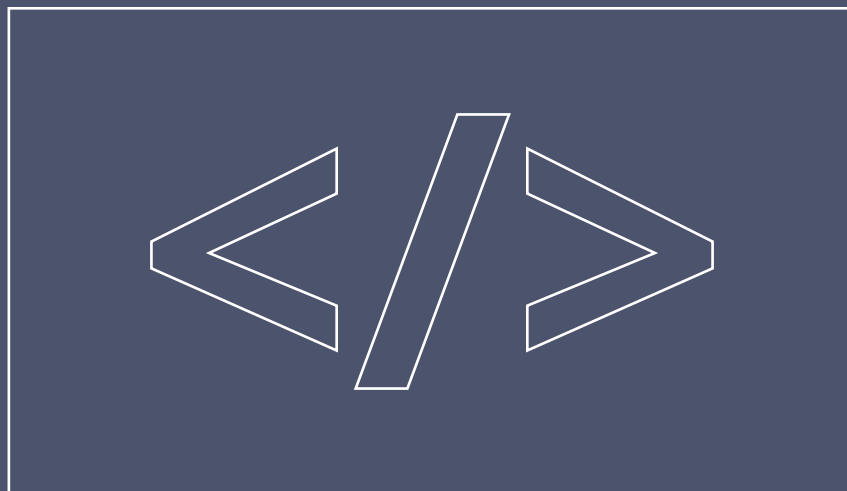




Incremental

Infrastructure as Code

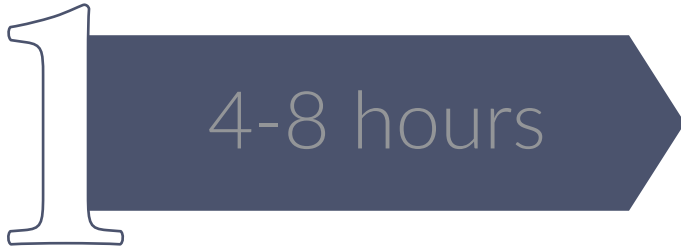
In just a few, short years, DevOps will become the new standard. Not because it's some disruptive revolution, but because it's an evolution that responds to the challenges of modern software development.

In this guide we'll cover *Infrastructure as Code*, and how to adopt it within your organization following our incremental approach. The approach we developed offers a sensible route for organizations to evolve towards DevOps.

The Incremental Approach

The incremental strategy is a six-step, cyclical approach that will not only minimize the risk and costs of Infrastructure as Code adoption, but build the necessary in-house skillset and momentum for larger and faster cycles. Each step builds upon the last, both in terms of investment and preparation, as will each overall cycle.

The hours are realistic for many enterprises, but will vary widely depending on the culture and politics of the organization. Meetings tend to consume the most hours, and these estimates assume a small implementation team (1-3 people) interacting with key server personnel (2-4 people).



Selecting a Suitable Server Role

The first step is a crucial one: select a suitable, small-scale server role.

If any teams or groups have expressed interest in tackling Infrastructure as Code, a role that impacts their applications would be a great place to start. It's likely the path of least resistance amongst personnel, and will maximize the likelihood of buy-in, cohesion, willingness to experiment, and ultimately, change.

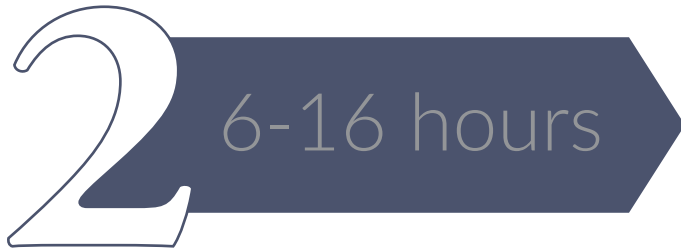
The initial server role shouldn't be technically overwhelming from a configuration standpoint, while at the same time it shouldn't be considered low priority by the business (otherwise, the value of Infrastructure As Code won't be visible), nor should it be the flagship server (too much perceived change at once). And most importantly, both the development and operations team responsible for the server role should buy-in and understand the importance of DevOps.

Server Roles: Defined

Server Roles help define common configuration between servers. They model the real-world overlap of servers, and provide the visualization necessary to easily understand your infrastructure. Server Roles facilitate the flexibility needed to address the complexities that modern infrastructure automation poses.

For example, you may have an iis-server role that ensures IIS is configured in a particular way, as well as hdar-api-host that ensures a particular web application is configured properly. Both roles can be applied to the same server so that hdar-api-host is configured on an IIS server.

Like applications, a single server role will span multiple servers across multiple environments (integration, QA, production), and are used in conjunction with the testing process.



Consider Short and Mid-Term Configuration Requirements

This step is generally accomplished in a single meeting, as the types of server configuration are already well-known by key development and operations personnel within an enterprise.

Although there's certainly no DevOps in a Box, the closest thing to a solution is a software product/tool, a process built around the platform, and a plan to implement both the software and process. The objective of this stage is to find solutions that would not just work for the selected server role, but that might, at least theoretically, work for other servers and servers roles in the organization. For instance, if the enterprise has a mix of Windows and Linux servers, then a using only a Linux solution should be a non-starter.

Server Configuration: Defined

Generally speaking configuration describes the packages, settings, files, or anything thing else that is on a server. For example, an Application Pool that's configured in IIS (along with any of the three dozen settings) would be considered configuration.

Just because the enterprise's servers are configured differently, there will be a lot of overlap in their configuration process. By the same token, just because two servers have similar IIS configuration, the process could be vastly different: for instance, a marketing website generally needs a lot less testing and scrutiny than stock trading API middleware.

It may seem like a lot to take in, but to reiterate, the goal is to find something that "might, at least theoretically, work." Now is not the time to craft a 20-page requirements document with a spreadsheet detailing each and every possible technical and process specification. There's simply no way of actually knowing each and every

server's configuration without spending significant time with each and every group to learn their technical requirements, so this initial organizational consideration must be done at a very high, almost "gut" level.

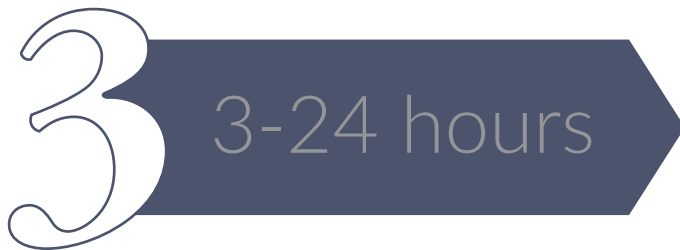
This step can largely be performed by the implementation team, although brief interactions with the development and operations teams may be needed to confirm technical requirements.

simply not worth taking the car home for the entire weekend.

Some tool vendors offer a self-guided tutorial that you can use after a download, whereas others will have salespeople give a live web demonstration. Both types of assessments should take about the same time (1-2 hours), and both should accomplish the same goal: not necessarily finding which tool will be "the one", but finding which tools will definitely not.

When doing a self-guided tutorial, make sure not to conflate the assessment with the next stage, the proof of concept. The assessment should be focused on finding an overall solution, not the server selected in the first stage.

To make this step easier, here's a comprehensive list of the top Infrastructure as Code Tools as of 2016.



Assess Tools

Tools are a critical piece of a DevOps solution and, by this stage, you should have a short list of them that seem to be suitable for both the selected server and the enterprise at a whole. Now's the time to dig in a little deeper to see which will be worth a more time-intensive proof-of-concept.

If you were shopping for cars, this would be the equivalent of a test drive around the block. If that's too bumpy of a drive, or you just really hate driving stick shift, then it's

- [Ansible Tower](#)
- [CFEngine](#)
- [Chef](#)
- [Inedo's Otter](#)
- [Puppet](#)
- [SaltStack Enterprise](#)



Proof of Concept

If the assessment was a test drive around the block, then the proof of concept is loading up the car with your family and pets, and then taking a road trip. At this stage, it's not just about finding the right tool, but discovering and defining the processes you'll use around the tool.

Because this will interact with the various servers that are in this role (and it's important to demonstrate actual value by to production, or as close as possible), there will be slight risk involved. The tools will need to be installed, firewall ports may need to be open, permissions and service accounts may need to be created, and so on. However, this slight risk comes with a fair reward: it'll give a feel for what is actually required to start automating your infrastructure and its server configuration.

To mitigate some of this risk, the proof-of-concept may be split into two phases: pre-production and production. Only upon completion of the first phase would the proof-of-concept be allowed to continue to the second.

An organizational change is as much one of hearts and minds as it is the tooling and processes. This stage will also give a feel as to the resistance and hesitation to change, as well as any perceptual leaps that need to be made.

Thus, it's equally important to involve and interact with the actual people who configure the infrastructure: testers, business analysts, network engineers, developers, etc. Just as the risk to automation can be mitigated with a multi-phase proof of concept, so can overcoming the resistance to change: only upon completion of one phase of the proof-of-concept would the solution be introduced to another person or team.



Pilot

The line between the proof-of-concept and the pilot can be blurry, but the phase generally shifts once the selected server's infrastructure is actually automated, and out in to the real world.

The pilot phase can be thought of as the “new normal” for this particular server. Everything has checked out, and the benefits weren’t outweighed by tremendous costs. Thus, by all accounts it makes perfect sense to continue using the newly minted devops process.

At this point, the new process is official: it’s time to license the tools appropriately and train staff appropriately on their new tasks and responsibilities. The only non-routine activity is to continually monitor and validate that the new solution is being well-received by all involved. And of course, share stories of success with other groups in the enterprise.



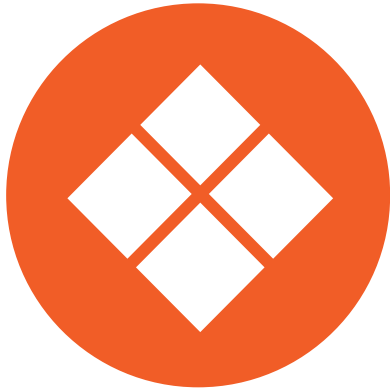
Iterate

In just a few months, the results of the pilot will be very real, measurable, and notable: the ease of configuration change, time saved, fewer headaches, etc. will all be reportable and transparent for other groups. And not just that, the business will start to see how much easier it is to automate changes faster.

From here, it’s back to the first stage: select another server. With all the experiences and lessons learned in implementing the process, each stage of the next iteration will be significantly easier, both technically and culturally. At some point, perhaps even after the first iteration, it’ll seem almost trivial to transition several servers to Infrastructure as Code in parallel.

Although this incremental process may take years, the results will be nothing short of transformative: the organization will have gained the knowledge and experience to not just implement devops, but to change and improve organizational processes. This experience will become invaluable as when moving onto new servers, especially when these new servers don’t even come close to fitting in the same mold as the other servers.

Remember, there is no end point of devops. It will continue to evolve as will how your organization addresses it. For now, just take what you have learned and apply it to other servers with an open, experimental mentality.



About Inedo

Our tools were built from the ground up for Incremental DevOps.

Release automation, continuous delivery, and server configuration all allow you to ship your software quicker, easier, and cheaper. We know getting there isn't as simple as flipping a switch. That's why we build tools that allow you to take a realistic, measured, and incremental approach to adopting these practices

Learn more about BuildMaster, ProGet, Otter or any of our special projects at inedo.com, and by following [@inedo](https://twitter.com/inedo) on Twitter.



Inedo, LLC
64 Front Street, 2nd Floor • Berea Ohio 44017
440.243.6737 • inedo.com

© 2016 Inedo, LLC. All rights reserved. BuildMaster, ProGet & Otter are either registered trademarks or trademarks of Inedo, LLC in the United States and/or other countries. All other trademarks are the property of their respective owners.