# Liatrio Principal Consultant Questions

1. **Q:** Can you travel up to 2 weeks out of 4 for consulting engagements to either coast? Are there any constraints we need to know about upfront that will prevent this regularly. (Mon-Thursday on site home every weekend)

* **A:** Yes and no :)

1. **Q:** If everything goes well, how much lead time do you need to join Liatrio and when is the earliest you can join. Also, post joining, do you have any PTO planned out already in the medium term future (6 months)

* **A:** I could be ready to join in a few weeks time. The only thing I’ve got scheduled right now is a trip to see family from 8/29-9/5 and DevOps World / Jenkins World on 9/18 & 9/19.

1. **Q:** How much time commitment can you provide for the interview process from the first call to final in-person interview.

* **A:** I’m available for an hour for each of the first few calls or video chats. If you expect they’ll go longer than that, I’m willing to work with you. If you decide to go forward, and want to meet onsite, that shouldn’t be a problem either. We should both have a pretty good idea by then if it’s a good match.

1. **Q:** Tell us how you have been involved in end to end software delivery that includes CI/CD, test automation, IaC, and Cloud

* **A:** I worked as an individual contributor build engineer for seven or eight years, starting at Microsoft then on to Envision Telephony, plus the first part of my time at Capital One. Since then my roles have been more “full pipeline”, adding more deployments with Chef and AnthillPro (not on my resume; I don’t want to use it again but am happy to talk about it ;)). My involvement with test automation is tangential. I’ve worked with functional test and performance test teams to integrate “their stuff” into “our stuff”. Same goes for unit tests, although we’ve been doing that for longer.
* As for IaC and cloud, I’ve been working with AWS for four years or so, have one certification (Developer Associate) and am working on a second (Solutions Architect Associate). I’ve mentioned elsewhere in here that I’ve been using Ansible for a while (Terraform and Fabric too). I like it as a tool, and being a fan of Linux and remote management via SSH[[1]](#footnote-1), also like how it does what it does. At the same time, “serverless” is carving out a niche and Docker with sane container orchestration may be a huge boon.
* People are doing interesting stuff with Docker and FaaS like Lambda, but I don’t have much visibility into enterprise adoption. We use Docker Compose here at Nortek for some of our developer environments, and I use Docker via Jenkins and Bash to spin up containers for unit testing in our CI/CD, but Docker doesn’t make it “up the stack” into our test, performance or production environments. We also use Lambda for some ETL work, where it’s a pretty good fit.

1. **Q:** Quickly define your familiarity with the following tools including when was the last time you were actually hands on with any of them:

* Jenkins, Maven, Gulp or Grunt, Selenium, Cucumber, Chef or Puppet or Ansible, Vagrant or Virtualbox, Nexus or Artifactory, AWS or Google or Azure
  + **Jenkins:** most recently used it to set up automated deployments in my current role; last used a few weeks ago
  + **Maven:** not much experience with it; I’ve built and packaged batch jobs at Capital One (ShareBuilder) to do end-of-day brokerage processing; last used close to two years ago
  + **Gulp:** no experience; **Grunt:** I’ve built and deployed NodeJS code at work for a partner application (“Dealer Portal”); last used a few months ago
  + **Selenium:** no experience
  + **Chef:** used a lot at Capital One for deployments which never seemed like a good fit; I mostly remember fighting with the central server; last used close to two years ago; **Puppet:** never for pay; **Ansible:** I’ve written a bunch of playbooks to spin up our environments as needed, and sometimes also use it for inventory queries; coached some of our developers on it, which worked out well; last used a month or two ago
  + **Vagrant** or **VirtualBox:** I’ve used both sometime this year; VirtualBox is currently running on my work laptop as a host environment for minikube (kubernetes)
  + **Nexus or Artifactory:** my team maintained an Artifactory instance for local deployments. We also used a shared (corporate) Nexus instance; last used close to two years ago for both (at Capital One)
  + **AWS:** very familiar with a dozen or so services; use it every day; **Google Cloud Platform** or **Azure:** never for pay

1. **Q:** Please define what you believe to be DevOps and what a DevOps transformation at a customer would look like.

* **A:** Great questions. I’ve started to answer it about eleven times already. My current answer (but there are others) is that DevOps is about putting responsibility where it belongs, and to a lesser degree, managing perceived risk.
* Broadly, I’ve always felt that developers like to change things all the time, while operations folks like to prevent things from changing. That imbalance used to look to me like the people with more information and less “skin in the game” were happy to roll out new changes then throw them over the fence, along with a partial description of what it is, how it works, and how to deploy it into a production environment. The operation folks took on the risk. If something broke, they’d be on the hook to fix the issue, often without enough background to do their jobs effectively. (That’s a concise description, but it lacks nuance. For the sake of this answer, I hope it’s enough.)
* A better outcome (for some reasonable definition of better) would have the developers deploying to whatever environment they have access to, possibly even production. But that’s just the tip of the spear: developers won’t suddenly have the skills, experience or confidence–or let’s face it, the interest!–to deploy to production. Getting there is a process, and DevOps works best when leading by example. In other words, when characterizing DevOps, even just on the topic of cultural transformation, there’s a lot more to it.
* In addition to ideation, design, implementation, pre-release testing, etc., developers are now starting to take responsibility for making sure it runs in its target environment. They’ll also respond to issues when they come up. The operations team took care of all that before, and the developers didn’t have to understand it. Now they may well have to rely on the operations teams’ expertise to get their work done. The specialists broaden their skill sets, and the team benefits as a result.
* The operations folks, on the other hand, have to become part-time developers. Until now, they could follow manual steps from a runbook, swapping in the specific configuration changes needed for a given release. But now they need to learn to write infrastructure as code, using whatever tools the team chooses. They may be excited! Or they may be nervous, lost or discouraged. That’s okay; they have a lot to learn. But if they were hired for their capacity to learn, rather than for some specific set of skills they’ve already learned, they’ll be fine. On some teams, the operations folks may have on-premises hardware, or other “pets”. Over time, they’ll be better served by moving to more dynamic environments, where resources can be provisioned on demand.
* So developers will have to take more responsibility for managing their applications across the full life cycle. And operations folks will have to take more responsibility for providing and maintaining environments into which to deploy and run those applications. And both will have share their sometimes siloed knowledge with each other to help the team succeed.
* I’ve not mentioned perceived risk, but it’s a big one. First virtual machines, then Git, now hosted cloud environments potentially lower to virtually nil the risk associated with making changes. Anyone can spin up an ephemeral EC2 instance and associated machinery (VPC, internet gateway, security groups, EBS, etc.) and have a live environment in minutes. With automation like CloudFormation or Terraform, they hardly even need to know how it was created. When they’re done with it, just delete the entire stack! This is a wonderful technical achievement, but I’m more interested in the way it changes peoples’ relationships to their compute environments. The “pets” are gone, and the “cattle” are slowly taking their place. No risk, no loss. Repeat as often as necessary, and please don’t think you need to keep some polluted old machine hanging around because “that’s how we’ve always done it”. Please. No.
* A successful transformation creates a stronger working relationship between the developers and operations folks. It allows for smaller and more frequent releases. All the code for this release and those prior is properly tagged in a trusted version control system[[2]](#footnote-2). It raises everyone’s confidence because changes are easier to understand and supported with unit tests. In addition, the environment is created as needed, and does not carry the accumulated detritus of past releases. The new environment is built from images that include the latest security patches. Monitoring and reporting is baked in, and dynamic. If the machine goes boom for some reason, rolling back, while almost always the last resort, is manageable.

1. **Q:** Now that you have defined a DevOps transformation, how would you execute this at a customer?

* **A:** A customer’s DevOps transformation means developers now maintain their creations throughout the full life cycle, including deployments. At the same time, the operations staff have evolved from operators to owners. But I’ve focused heavily on the cultural side so far. Time to get tactical.
* I’d venture to say that many teams (or customers if you prefer) are struggling with similar issues. I’d also suggest they have a couple things that make their situation different than others–unique being too strong a word here. At the same time, DevOps practices should be convergent and frankly boring. I want consistent, repeatable and no surprises/drama. In other words, I want to take a team from high to low variability, specifically with respect to the machinery that allows them to focus on what makes them unique. Very few teams, aside from CI/CD vendors, etc., has ever differentiated themselves in the marketplace by how they compile their bits!
* Describe for the leaders and the practictioners (developers and ops folks) what the future can be. Show possibilities. Maybe do a demo of a CI/CD flow or spinning up a new QA environment on the fly. Answer questions. Try (but not too hard) to respond to push back and recalcitrance. Show them another way of thinking about things. Don’t try to solve all their problems. Just plant a seed. Some will take hold, some won’t. That’s fine.
* Follow that by sitting down with the leaders to find out what the business needs. Focus on issues of velocity, quality and process. There will probably be more than enough there, if they’re bringing someone in to try and fix things.
* Then sit down with some of the practitioners. Can you get them to talk? Come with specific questions: what works with your software delivery process today? What doesn’t work? How long does it take to push a hotfix? A release? What’s your agile process like? Is it effective? Just get them talking. Try and identify those who are eager to be involved, and those who will take some time to come around.
* Think big! Start small!
* You now have a better idea of the issues facing the team and their collective mood/attitude. It’s pretty clear you intend to change some things. They’ve got work to do, and this ain’t it. But the real issue is you’ve got basically zero credibility with the team right now. You need a quick win. It doesn’t have to be much, but it absolutely should make people understand how using it will improve their situation, even if just slightly. There may be resistance, especially if you want to touch something that people don’t touch (it works, just leave it alone). Don’t choose that thing, but find something. :)
* Assuming that trial effort went okay, now is the time to engage with those who are excited about what they’re seeing! They’re your MVPs. They’ve got credibility (hopefully ;)) within the team, and they’re open to what you’re there to help them do. It may not be feasible, but if you can, work with the leaders to put a mixed team together, and identify a more substantial project to tackle next. Regardless of what project is chosen, it’s crucial that you’ve got talent on the team to help move it forward.
* There may be a bunch of administrative tasks to take care of, as people wind down their sprint work, maybe set up a temp distribution list on their mail server, schedule meetings to start planning, etc. But this is a critical step. If you’ve gotten people excited about what they’re doing, they’ll give you their best.
* Somewhere in here you’re going to run into a bigger challenge. You may not have much in the way of infrastructure already built out. You’re going to need to bring in some “practitioner consultants” to guide the members of the new team. While the DL and everything is getting set up, you may have time to pair up some of the “practitioner consultants” with new team members for individual coaching.
* Things are heating up now. It will hopefully snowball on you. But I’ve gone on long enough (probably too long) for now. Thanks for causing me to reconsider these types of questions!

1. **Q:** Have you led a DevOps transformation and if so:

* Please provide a specific company name/scenario AND a key accomplishment during that project?
* Why do you think this accomplishment was significant for your client? How about your company?
* **A:** One of the last efforts I was involved in at Capital One was moving from our legacy version control system (AccuRev) to Enterprise GitHub. We had eight weeks to get it done. My manager and the VP who ordered the work did an excellent job of communicating the business need to make this transition. Myself and our project manager at the time managed the bulk of the organization, reporting up and down, metrics collection, etc. We worked with about twelve different sprint teams to move all sorts of different depots (AccuRev’s word for something like a repository) and streams (similar but not really to a Git branch) into GitHub.
* As well, for some of the Java batch jobs, I wound up doing the conversions myself because the teams were somewhat gutted at the time. (This all took place in the middle of layoffs and re-orgs.)
* Anyway, we ran into some interesting issues with reporting. It was very difficult to accurately report at the right level of granularity, because one depot might have one stream, and another might have many streams. Or the source from one depot might be a simple conversion, say Gradle to Gradle, but another one might be some kind of homegrown thing that needed rehab in order to work on the other end. I don’t recall the details, but do remember being stumped on how to report things such that they didn’t look too rosy when in fact they were still in heavy development.
* The key accomplishment was we shut down the contract for AccuRev (the artificial but effective reason for the urgency), saving a bunch of money. I don’t recall the dollar amount today, but it was at least $100k/yr., if not more. From a developer perspective, having our code colocated with the rest of our organization (Retail and Direct Technology) was also substantial.

1. I’d love to evolve to truly immutable infrastructure, but at the moment consider it more an ideal than a practical reality for many teams. [↑](#footnote-ref-1)
2. Side note: It’s my view, maybe not widely shared, that a planned release and the response to an outage incident are similar to one another, and should be treated similarly in many respects. In both cases, the code to deploy to add a new feature or fix a bug live in a trusted repository. The steps to deploy should be conservative in that what works for one should work for the other. [↑](#footnote-ref-2)