Operational – the forgotten architectural view

Eoin Woods

Most books on software architecture focus on building new systems. However successful systems spend much more time running in their production environment than being initially developed. This is why the recent emergence of the "DevOps" movement [1] is so heartening, with its emphasis on getting development and operational staff working together as early as possible and sharing tools, processes and practices to smooth the path to production.

Embracing DevOps involves embracing lots of new technologies and ideas and can cause a lot of confusion for many people involved. Architectural thinking and design can help to clarify the stakeholders involved, their concerns and how those concerns are being met.

There is a recent book on DevOps technology and practice aimed at software architects [2], but beyond that, there isn't too much to guide architects in dealing with the operational environment for their systems. In this column I'll outline an architectural viewpoint that tries to do just that. It's an update and reworking of the viewpoint of the same name from [3].

What Is "Production"?

Before we go any further, it is worth briefly defining what we mean by "production" as it's a widely used term whose meaning can vary. In this context we mean any environment which is being used to perform valuable work. This also usually implies that the environment is a "controlled" environment in that it can only be changed in accordance with some sort of change control policy, rather than directly by developers. Will anyone care if you remove the contents of the database by mistake? If this is a disaster then you've probably got a production environment!

Characteristics of production environments that make them difficult to work in include:

- A wide range of stakeholders who have an interest in its operation, including business management, auditors and risk managers, infrastructure and operational staff, as well as the normal users and development staff who are also interested in the development environment.
- A high degree of control, which is intended to contribute to the reliability of the environment, but which means that there are usually significant processes associated with making a change.
- A high degree of visibility, particularly when things go wrong. Few people are really interested in the status of your development environment, but when production environments malfunction you quickly find out just how wide your stakeholder community really is!
- Vulnerable to external events, particularly when connected to the Internet. Your development and test environments are probably hidden away, your production environment is there for anyone to see.
- Unpredictable due to being part of a much more complex environment than most development and test systems, leading to it being affected by external events that you may be unaware could even happen.

All of these factors combine to mean that working in the production environment can be quite challenging and as software architects we need to address these challenges as part of our work.

Sidebar: Recapping Viewpoints and Views

I think the idea of an architectural view and its corresponding viewpoint are widely understood today, but it is worth briefly recapping what they are. A view describes one or more structural aspects of an architecture, in order to show how the architecture addresses one or more concerns for one or more stakeholders. A viewpoint is a guide to creating a particular type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing views of this type. So our Operational viewpoint is a guide to creating Operational views.

The Operational Viewpoint

An Operational view describes how a system will be installed, operated and supported in its production environment. The deployment environment that the system needs (servers, software, networks and so on) is addressed by the Deployment view [4]. The Operational view is usually applicable to any system being deployed into a complex or critical operational environment.

Key Stakeholders

When considering the operation of a system, we often just think about systems administrators but there are a wide range of people who have an interest.

- Operations Staff, who accept new and changed software into production, operate the system in production and are responsible for its service levels;
- Infrastructure Engineers, who are responsible for providing the infrastructure services the system relies upon.
- Developers, who are responsible for the software, its smooth transition to production, and ultimately accountable for its success.
- Testers, who need to be able to verify that the software will operate correctly in production and that the production environment as a whole will operate correctly.
- Communicators, who in the context of developing a product for installation on client premises, need to explain the operation of the system to clients.
- Assessors, who need to be satisfied that the risks of operating the system in production are acceptable and managed.

The needs of all of these stakeholder groups need to be addressed, not just Developers and Operations.

Concerns

The key concerns that these stakeholders tend to have about the operational environment include:

- Installation and Upgrade – how are software changes going to get to get to the production environment reliably? How will you know that they have been applied successfully? How will failures be rolled back?

- Migration – a related concern is that of migrating workload to the new software and performing any changes needed to the system's data (both the storage schemas and the data itself). Will multiple versions of the system exist in parallel or does everything migrate at once?
- Operational Monitoring– once the system is running in the production environment how do you know if it is operating correctly? How do you control the system's operation? What "vital signs" need to be monitored? Some are likely to be very standard (such as CPU usage) while others will be very specific to your environment (such as message volume received on a particular interface). Business measurements (such as average and total transaction value per hour) are likely to be just as important as technical metrics.
- Operational Control - what tools do you need to control the system? Can you do all this with third party tools or do you need some of your own?
- Alerting – if the montoring mechanisms identify an unexpected condition what should happen next? How does this event get recorded and propagated? To whom? And what do they do? How will you use the history of monitors and alerts to provide continual improvement to key metrics?
- Configuration Management – most modern systems are dozens or hundreds of infrastructure elements, some of the largest comprise millions, and it is becoming common to add and remove virtualised infrastructure elements on demand. How will you manage the configuration of these elements? Modern tools like Puppet [5] and SaltStack [6] can help to simplify the process but are only part of the solution and may have an impact in how you design and deploy the application.
- Performance Monitoring – well documented evidence [7] suggests that as Internet applications slow down, users leave. In-house users are more tolerant but performance is still often one of the biggest factors in their satisfaction with a system. How will the application's performance be monitored? What are the important metrics to measure? How will degradation be spotted?
- Support – things inevitably go wrong in production environments, usually in highly unexpected ways! Who will handle the incidents that occur? What tools and processes will they need? How can the application be designed to be easy to support?
- Data Availability – backup and restore is an obvious concern that is as old as computing, but one that needs more thought that it is often given. With today's huge databases, backup and restore can become mammoth operations unless very smart strategies can be used for them.

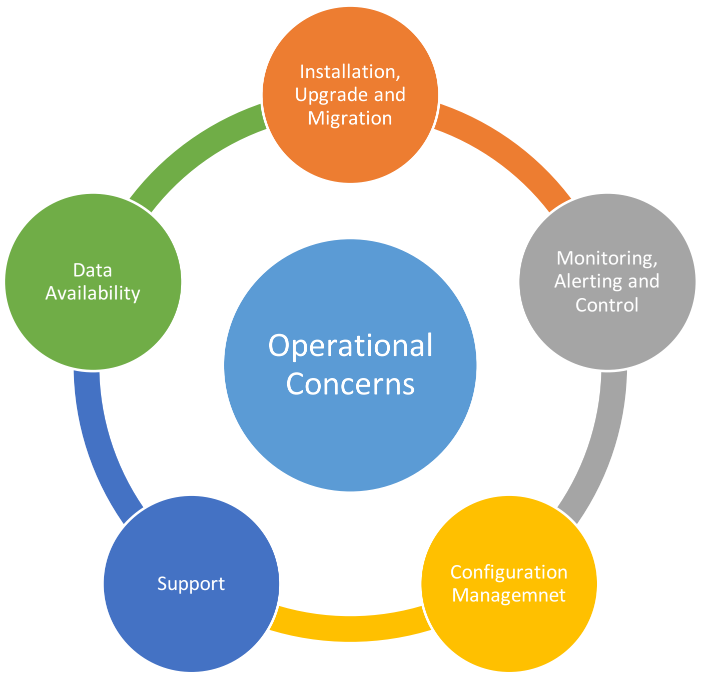

Figure 1 - Operational Concerns

Models

The essence of architectural work is understanding and solving problems, which often involves creating models to understand systems and support good decision making. I don't have space here to describe in detail the types of models for addressing operational concerns. However, in outline, some of the models that I have found useful have been:

- Release Models, which describe the "path to production" from the development environment (in terms of stages, technologies and approval checks). In effect a model of the route from the end of your continuous—integration pipeline, to the production environment. This allows you to communicate it clearly, identify risks and weaknesses,
- Configuration Management Models, which help you to capture and analyse the different types of configuration that you need across your operational environment and how it will be managed and controlled. Today we can easily end up with configuration in properties files for Java software, application settings found in databases, Zookeeper for distributed systems and Puppet for infrastructure, so understanding, coordinating and validating change across all of this is a major challenge, which some focused models can help to meet.
- Administration Models that show how the administrative environment relates to the system and how it works. Administrative environments are often a complex mix of standard tools, local utilities, people and processes and to avoid lots of problems and misunderstandings, a model of how you believe it's all going to work will be of great value when working out how the Operations group run the system.
- Support Models, which show how an incident is recognised, handled, managed and resolved. These models are usually process descriptions rather than technical designs, but they're very useful tools to allow various types of problem scenario to be considered so that you know you've how each will be handled.

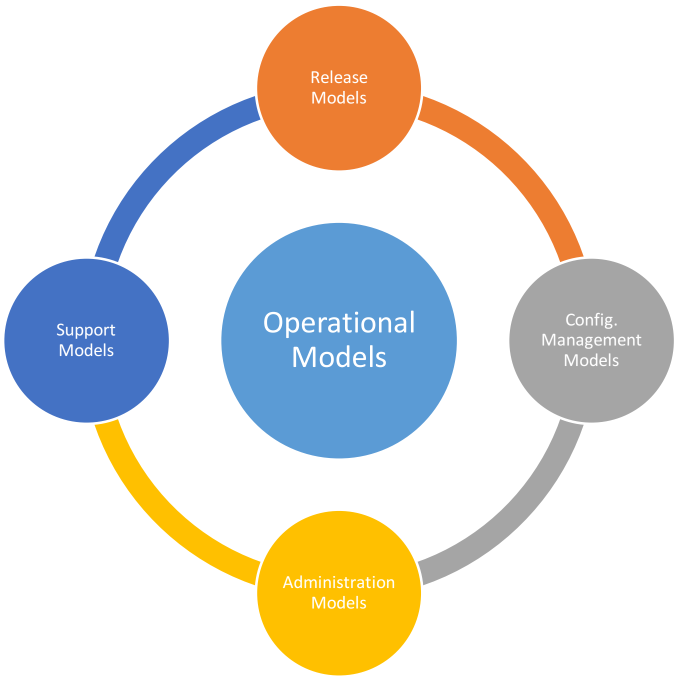

Figure 2 - Architectural Models for Operational Concerns

The thing to recognise about these models is that they're not typical software architecture or design models, they're pragmatic models focused on particular concerns. So you'll usually need your own "boxes and lines" or "text and tables" approach to creating them. They also need to be developed in conjunction with the Development and Operations groups to make sure that everyone has a common understanding of how the operational environment will work.

Problems and Pitfalls

Some of the specific things that can go wrong when working on the production-oriented aspects of your architecture include:

- Late or poor engagement with the operational staff, meaning that while you might be using lots of "DevOps tooling" its going to have very little effect because the Operations group don't buy into it.
- Lack of backout planning. A lot of discussion about continuous delivery and DevOps is about how you get code to production safely but not what to do if things go wrong and you need to rollback. Blue/Green Testing and Canary Releases are powerful approaches [8], but they aren't magical, and it can be very difficult to rollback quickly with a large database after a high impact database change.
- Lack of migration planning, which is often as much about people as the technology. Is everyone moving on one day? Do you have a pilot phase? Can you Blue/Green deploy? How do you know if things are working for particular user groups? How will data get migrated into new databases or schema forms? Is there a realistic timeframe for all of this?
- Missing management tools or processes, which can be a symptom of not engaging early and often with Operations. What are you assuming they'll be doing? Do they have all the tools they need?
- Poor alerting, often caused by wanting to ensure a high degree of visibility, but not thinking through failure scenarios. It is very easy to create a tsunami of alerts and

this can make it impossible to understand and address the underlying problem. Is your alerting smart enough to allow Operations to pin point the underlying problem quickly? (An important question for all of your monitoring tools vendors!)

- Lack of integration into the production environment, which again is normally caused by late collaboration with Operations. What are they expecting in terms of tools, documentation, processes and so on? If you don't get this right, you're probably not going live, and these factors are often difficult to judge in pre-production environments.
- Inadequate backup and restore strategy. In particular if you have a catastrophic failure in networking or a major data centre problem and need to operate from another location, do you have all of the data you need there? Do you need to perform any restore operations before commencing? And if so how long do they take?

In Conclusion

Systems exist to run in production and deliver value, but sometimes when you look at most our software architecture literature you'd be forgiven for missing that, as it's often not emphasised. The DevOps movement is a terrific step forwards, emphasising the importance of the Operations group and the need to unite development and operational tools and processes. Software architecture still has a big contribution to make as software moves towards production as we still need to understand the stakeholders involved, their concerns and how we address them. Thus architecture work can guide DevOps work to adapt and focus within in a particular situation. In this column we've outlined the Operational viewpoint that aims to provide the architect with a guide to achieving this.

References

[1] G. Kim, K. Behr and G. Spafford, The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win, IT Revolution Press, 2014.

[2] L. Bass, I. Weber and Z. Luming, DevOps: A Software Architect's Perspective, Addison Wesley, 2015.

[3] N. Rozanski and E. Woods, Software Systems Architecture, Addison Wesley, 2011.

[4] N. Rozanski and E. Woods, "The Development Viewpoint," [Online]. Available: http://www.viewpoints-and-perspectives.info/home/viewpoints/deployment.

[5] Wikipedia, "Puppet (software)," [Online]. Available: https://en.wikipedia.org/wiki/Puppet_(software). [Accessed 03 03 2016].

[6] Wikipedia, "Salt (Software)," [Online]. Available: https://en.wikipedia.org/wiki/Salt_(software). [Accessed 03 03 2016].

[7] B. Forrest, "Bing and Google Agree: Slow Pages Lose Users," O'Reilly, 23 06 2009. [Online]. Available: http://radar.oreilly.com/2009/06/bing-and-google-agree-slow-pag.html. [Accessed 03 03 2016].

[8] D. Farley and J. Humble, Continuous Delivery: Relible Software Releases Through Build, Test, and Deployment Automation, Addison-Wesley, 2010.