Notes - Organizational Development - Technology - Part 1

- Dr Nick Hayward

A brief introduction to technology choices and structure with example usage for software projects.

Contents

Intro

Technology choices, both initial and ongoing, are a key part of managing and organising a project group, department or company.

Whilst technology trends may come and go, evolving rapidly to meet new requirements and market demands, there is also a complementary consideration of legacy, ongoing, and *tried and tested* solutions.

Production level systems may often last many years and cycles without significant modification and change. Such systems may endure for various reasons but they require consideration and attention as much as new and upcoming solutions and products.

For example, we might see long-term production systems for reasons such as

- slow release cycles for current production systems and technology stacks
  - e.g. due to requirements, features, logistics &c.
- company management may not wish to modify or potentially break their main revenue stream
  - i.e. if it's not broken, don't fix it…
- perception that the current production system cannot be improved or updated
  - a lack of incentive or motivation to improve or update
- management and technology support does not have appropriate experience beyond current technologies and stack
- …

Upgrade considerations

When the time comes to consider an upgrade or migration to a new system, we might consider the following

- how will the new, updated system itself evolve over the coming years?
- is there a backup plan if the technology becomes deprecated or end of life?
- are there any specific required dependencies for usage? considerations for longevity of such dependencies &c.
- human factor for support of such technology, e.g. programming language or systems experiences specific for the technology &c.

Recent Windows based systems are a good example of deprecated support, and the need for long term considerations. For example, updates to Windows 10, and now Windows 11, provide a virtual machine

for limited support of legacy, deprecated applications. These include, for example, required applications originally coded with Visual Basic, and various third party plugins.

There are many similar examples of deprecated software and products requiring new solutions. For example, the formal *W3C Recommendation* status of HTML 5 in October 2014, along with public support from companies such as Apple for the media element `<video>` , provided an opportunity to remove and replace the much maligned Adobe Flash as a viable embedded video player for web based content.

tightly coupled A tightly coupled, or locked, technology dependent system should be avoided where possible.

Regardless of the specific context, from an application to a department, we should try to avoid unnecessary reliance on a specific vendor, programming language, or framework.

The key question becomes whether such a decision restricts the ease of updates and corrective modifications when required by changes in technology.

For example, if a vendor or framework ceases development, stops supporting a given product, how easy will it be to switch out thst technology or product for something equivalent.

With the best intentions, such decisions can cause issues and additional costs, both maintenance and financial, at some point during the lifecycle of an application, project, or company.

A common cited justification for avoiding initial abstraction and lightly coupled developemnt and technology implementations are the underlying overheads in planning, structure, development, knowledge and experience &c. required for initial implementation.

The financial costs to work with such considerations, compared with a vendor or service lock-in, may be too high for many companies to consider. It may simply be easier to follow the path of least resistance with a vendor supported pipeline from start to finish.

Licensing costs, or the potential for future increases, should also be a consideration when choosing a framework or toolset for development.

The programming lanaguage *Java*, for example, has been a popular option for many years with a wide array of companies and markets. As a programming language, it offers many options for application development, portability across varied systems, implementation and setup, &c.

For example, the choice of Java Virtual Machine (JVM) provides an opportunity to abstract implementation whilst choosing a specific option to solve a given developmental or structural problem. Various alternatives are available, which may be implemented without fundamental modifications to the original application or project code.

Deployment of the language is supported across multiple operating systems (OSs), and environments from Linux to Windows, and almost everything in between as well.

When making such decisions, it's usually beneficial to consider and create multiple opportunities for exiting or removing a given technology from part of an application, project, or even at the company level.

custom build or acquire A common consideration at many levels of project development and management is the trade-off between acquired or purchased technologies and services with in-house custom solutions.

There is no definitive answer for the option, and it will depend on many internal and external factors.

For example, the sheer scale of the group or company can create disparate issues for such technology decisions. As a a project or company grows in complexity and structure, the technology requirements customarily increase to match. It may no longer be viable to simply purchase and deploy an off-the-shelf solution.

When the decision is made to follow a *build* option and strategy, the usual issues become apparent, including financial costs, knoweledge requirements, time constraints, deployment and maintenance &c.

However, there is also a consideration of intellectual and research costs, which might help justify such choices. For example, we should not be redoing or rehashing existing products and technology. Reinventing the wheel is rarely beneficial beyond mere educational goals and aspirations.

As such, we may need to consider the potential for intellectual property (IP) gains from such research. This may help offset some of the initial perceived costs for developing a custom, in-house solution.

If a purchased solution is preferred, we should also consider and review associated costs for initial setup, ongoing licensing fees, hosting and maintenance, where applicable, ongoing educational costs for new personnel training &c.

make a decision Regardless of the final decision, develop or acquire, the team manager and company need to be able to justify and explain their decision.

For example, how might a project, department, or company explain and justify to other stakeholders a high initial cost for custom development compared with simply purchasing an existing solution.

How might a project leader or manager clearly articulate the downside of ongoing costs, maintenance, and the issue with tightly coupled systems.

Ongoing maintenance of decisions, records, performance &c. will help provide supporting material, evidence for such decisions. A simple pros and cons list may suffice for such explanations and discussions.

Services and cloud

Cloud based services, including hosting, applications, APIs, datastores &c. have continued to grow in popularity in recent years.

Such options have also grown in general acceptance amongst many companies, including existing technology developers and providers.

What are some of the differences and defining characteristics for *cloud* versus in-house hardware solutions. For example,

- cloud - usually billed as a per usage service, commonly providing CPU, storage, memory, and network options on a case-by-case basis. Various setups will usually be offered, providing a mix of pre-boxed solutions and bespoke systems without specific consideration of physical hosting hardware…
- in-house - the hardware required for servers and service provision, including CPU, storage, memory, network &c., will be stored, managed, and maintained by the company itself…

perceived issues However, whilst it may seem that the *cloud* is almost ubiquitous in its adoption, you may still hear some of the following issues with deployment and reliance on cloud based solutions.

For example,

- security issues - a persistent comment regarding cloud storage, but one that improved significantly in recent years. This includes improvements in automated backups, silos and network isolation, broader adoption of encryption by default, containers, isolated and fine grained interfaces and rules, security first approaches to data access…
- general costs - commonly perceived as expensive to ruinous in the wrong use cases and contexts, this has been addressed in many cases with fine grained access limits, quotas for billing, reduced network access and performance for lower costs, &c. Such costs, however, need to weighed and compared against in-house solutions, included associated replacement, labour, security, insurance &c. costs for dedicated hardware options.
- data access and permissions - company and personal data will be lost to the service provider and host. Whilst this may seem like an obvious issue, it may, of course, be mitigated with correct selection of hosting location, local data protection laws, encryption, data permissions &c. Whilst there are associated network costs for data access, this is, in most cases, offset by the inherent benefits of cloud based services and hosting.

perceived benefits Cloud services, hosting, and general computing is, however, more than simply data storage and static hosting.

To gain tangible benefits from the underlying nature of cloud solutions and services, compared with traditonal, in-house hardware provisions, we may consider the following as a sign of cloud benefits.

For example,

- elastic nature of performance, scale, access &c. - if a company's cloud bill remains fixed or predictable per month, this may be a sign they are not leveraging the benefits of the underlying *elastic* nature of cloud providers. Most companies will experience highs and lows for access to their services, products, and data. This should usually be reflected in the associated use of cloud options.
- rolling releases, CI/CD &c. - modern cloud methodologies and practices encourage a container-like way of approaching updates and product releases. In effect, if a development or engineering team requires server downtime per release, they may still be using servers in a node-like, static manner. Instead, they should adopt the cloud-based approach replacing servers with each release, and migrating users and clients as tests and validation meets defined requiremnts.
- load balancing and scaling to meet requirements - cloud based providers and services have long touted the inherent benefits of quick and easy scaling to meet variations in load, performance, costs &c.
- data storage limits - cloud storage and services theoretically provide an unlimited pool of resources and storage based on an incremental payment model. In effect, if you can afford extra storage, you can usually add it to a cloud account and system.
- reduction in data loss - whilst possible for cloud based storage and services, the potential for data loss is usually mitigated by a choice of automated backup, redundancy, and elastic infrastructure options from the chosen cloud provider.
- hardware errors and faults are reduced - whilst suitable backup and redundancy options should be considered and provisioned for a cloud based service or app, the potential disruption from hardware failures is, customarily, reduced with a reliable cloud provider. However, such provisions and failsafes, as with traditional in-house server and hardware solutions, should still be added to a cloud based solution.
- reductions in custom, in-house server solutions - network management, fault tolerance, load balancing &c. are usually provisioned and maintained by cloud providers, thereby reducing the need for custom solutions and products.
- error and disaster recovery with built-in solutions - cloud based services and providers will usually provide built-in solutions and options for seamless disaster recovery, instead of the traditional in-house *hot swap* provisions. This might have included replacement servers, array striping and raid solutions, backup recovery &c.

A notable benefit, if implemented correctly, is the use of containers for app and service provision. This includes various potential security benefits such as

- container isolation - whilst not guaranteed by default, containers do inherently prevent the invasion of malicious code from affecting other containers or the host system, thereby enhancing network isolation. (IBM)
- lower operational costs - container security strategies can significantly reduce operational costs. The cost to fix an error found after product release is much higher than one uncovered during design. (Cloud Security Alliance)
- elimination of perceived blind spots - adopting a resilient container security strategy can help eliminate risky blind spots in the network. (Cloud Security Alliance)
- improved visibility - cloud services providing container solutions help reduce and overcome some of the perceived limitations of vulnerabilities with self-hosted and managed Docker containers and local open source server software.
- …

Choosing the right options

For project teams and companies accustomed to traditonal, in-house server options and solutions, the cloud can seem a tad overwhelming and difficult to navigate. In effect, a myriad of options, server types, data stores, backup and recovery options, &c. This does not even include the various payment plans and usage quotas that often need to be considered and managed for a successful cloud based solution.

However, there are a few initial considerations we might review for cloud based solutions.

For example,

- use lessons learnt from in-house provisions to select the best cloud based solution, i.e. how can I begin to replicate existing server solutions? This will commonly provide a useful starting point for further consideration of cloud options and solutions.
- it's also useful to remember that security provisions, whilst potentially improved with cloud options, are not guaranteed, in particular as a default position for an application or service.
- issues, failures, and server downtime will happen regardless of the best laid plans and infrastructure considerations - with this in mind, it's important to remember that the cloud, in and of itself, is not an error free environment. Problems will arise, and a plan should be devised, practiced, and created to handle potential failures with server and cloud based options. If the underlying architecture for a cloud infrastructure is well considered, with appropriate failsafes and redundancy, errors may be seamlessly handled to provide minimal and, hopefully, zero disruptions for clients.
- a knock-on effect of such preparations, if not considered with due care and attention to resource management, is excessive provision and consumption of resources. The cloud may also provide scenarios for production hosting, management, and releases. However, whilst many cloud providers will gladly provide such custom options, based on custom environments, it is also easy to destroy and teardown such production hosting, databases &c. For many of these services and cloud based solutions, there is still, rightly, a key inherent assumption that the developer, admin &c. is aware of the underlying options and systems, including potentially destructive commands. In effect, an underlying assumption that developers, account owners &c. know what they are doing with appropriate prior knowledge.

The cloud has many inherent benefits, as noted above, for hosting, production releases, services and interfaces &c., but many of the traditional resource management considerations will still be appropriate per project. The key difference is that some of these underlying issues have now migrated to cloud providers and services.

failure built-in Cloud based apps, for example, will commonly include built-in failure handling and management. A single failure in a provisioned node, including server, database &c., should not bring down the whole system. In effect, such failure management is part of the structure and underlying nature of cloud native apps. This is a common expectation and template for a good, working cloud native app.

Resources

- "Design your organization to withstand future disasters" - Harvard Business Review - https://hbr.org/2022/03/design-your-organization-to-withstand-future-disasters
- "Designing disaster recovery strategies in a software product" - Rootstack - https://rootstack.com/en/blog/designing-disaster-recovery-strategies-software-product/
- "Disaster recovery plan examples (with how-to steps)" - Indeed - https://ca.indeed.com/career-advice/career-development/disaster-recovery-plan-example
- "Disaster recovery plans for IT ops and DevOps pros" - Atlassian - https://www.atlassian.com/incident-management/itsm/disaster-recovery
- "Gartner Research and Advice for Disaster Recovery" - Gartner - https://www.gartner.com/smarterwithgartner/gartner-research-and-advice-for-disaster-recovery
- Wallace, Michael and Webber, Lawrence. "The Disaster Recovery Handbook." 3rd Ed. 2017. O'Reilly. - https://www.oreilly.com/library/view/the-disaster-recovery/9780814438770/
- "What is a disaster recovery team: roles for a successful plan" - Tierpoint - https://www.tierpoint.com/blog/disaster-recovery-team/