Notes of an IT Architect

Автор:
Eugeny Shtoltc

Notes of an IT Architect

Eugeny Shtoltc

In this book, the Chief Architect of the Department of Architecture and Management of Technical Architecture of the Cloud Native Competence Center of Sberbank shares the knowledge and experience with readers, accumulated in the development of their own and assessment of other people's architectures, providing a basis for professional and career growth.

Eugeny Shtoltc

Notes of an IT Architect

About the book

In this book, we will cover the following sections:

* Architecture;

* Solution Architect and microservices;

* View from the height of the business and business architect;

* Corporate architecture;

* Service architect;

* Use in the management of ITIL 4, PMBOOK and COBIT 5;

* Application architect and design patterns;

* DevOps as a component of an architect;

* Architect and basic patterns;

* Corporate data bus;

* Service Oriented Architect;

* Applications in the cloud;

* Infrastructure for the cloud;

* Edge scaling sizes: data centers, cluster, sizes;

* Architect in business processes;

* Waterfall;

* Scrum;

* Kanban;

* Varieties of teams;

* Selection and growth of personnel;

* TeamLead & leading specialist;

* Virtualization;

* Features of development in Windows – Vagrant;

* Containerization;

* Podman and Docker;

* Stacks;

* Languages and paradigms of programming;

* Front-end: single page web applications.

About the author

Now the author holds the position of the Chief Architect of Cloud Native Competencies of the Architecture Department of Sberbank Competence. In this position, the author is engaged in research on the implementation and use of technologies that are already or very soon will become the de facto standard, such as servoless theologies and CloudEvents, and with which he shares with the reader. Also, the author, on a regular basis, evaluates existing systems and those planned for implementation in accordance with their modern standards, for example, CloudNative. About this, in the book, the author talks about the scope and guides the reader to implement them. The author pays special attention to this – finding practical solutions in the formed ecosystem that make life easier for both developers and the support service, while the customer remains interested in them. Employees gain knowledge of current technologies in which problems that are present in obsolete and retired ones have already been solved. This solves the problems of both developers and customers. The author does not dwell on any one technology that he has come across, but gives universal technologies in a systemic and practical form, or introduces the reader to the set of used ones, as the author gives the code in the languages of Go, NodeJS, PHP and Java depending on the relevance. more than 10 years of experience in various fields and in various positions allows us to highlight the relevant and popular ones, as well as undergoing training at Yandex, Sberbank EPAM and receiving many specialized certificates. The author has collected experience, both in domestic and foreign companies, both in startups and in enteprase, both creating his own co-commercial products and working in grocery development and outsourcing, producing streaming products, as well as complex proprietary software solutions… In addition to typing systems and practical help, the author gives an organizational minimum. Internally, the vision will enable work experience in various technical positions such as back-end and full-stack developer, DevOps and Team-Lead, including Software Architect. Experience of work not only as a hired employee allows us to take a look at the organizational level. The author worked both as a hired employee, but also as an individual entrepreneur and an official partner of a large supplier of mass software in Russia and the CIS (Bitrix Technology Partner), making and assembling customized and scalable solutions.

Architecture

GOST R 57100-2016 (docs.cntd.ru/document/1200139542) based on the international standard ISO / IEC / IEEE 42010 defines architecture as "Basic concepts and properties of a system in the environment, embodied in its elements, relationships and specific principles of its project and development". There are quite a few varieties of it, but we will highlight the main ones in terms of abstraction level: Application Architecture, Software Architecture, Solution Architecture and Enterprise architecture. An application architect develops the architecture of the application itself using design patterns and task allocation, and often combines his role with the role of Team-Lead or Lead Developer of Responsible Components (Tex-Lead). Software Architect does the same thing as an application architect, but works with multiple teams to add unification to the technologies they use. This position is often in demand in outsourcing, where there are many projects and there is an opportunity to take the load off Team-Lead so that they communicate more with customers and the team. This position is characterized by requirements for a vacancy in knowledge of the programming language and the main stack used on projects. In such a situation, the architect is limited in his choice of technologies and hiring new employees. Since its inception in 1959, the architect has dealt with the decomposition of the system, the distribution of parts among the developers, and was responsible for the subsequent integration of the developed components into the originally required system. Now the situation has been simplified with the advent of microservices.

An enterprise architect designs interconnections between systems using an enterprise data integration bus, and an application architect designs the systems themselves, decomposing them into applications. The boundaries between applications are determined by the boundaries of use: development, deployment, provision to the vendor. Previously, applications were also combined by technology platforms and technologies, but with the advent of containerization, provisions can contain components created on different platforms, languages and stacks, enclosed in containers. It has also lost its relevance to the formation of boundaries based on the deployment of the application due to the fact that components (containers) are rolled out and are already being tested in the environment of other components. Ideally, a group of micro-services is grouped by function performed by the business and the team that develops it, but often business processes involve common components, which blur the boundaries of applications. This specificity has led to the emergence of a separate specialization – Cloud Solution Architect.

Based on the level of architecture that is supposed to be designed, it is possible to turn from an abstract question – how to become an architect – into a set of requirements necessary for solving a given problem: from purely technical to organizational. So a software architect can delegate all organizational activities to Team Lead and focus purely on the technical description of the structure of the program, and often he is a pure techie and also Tex-Lead, but cannot delegate the technical one in any way. In contrast, the corporate architect can be a non-technical specialist, for example, a director, conducting communication to organize the connections of automated systems and satisfy these systems to the needs of customers. Based on this, one can guess that when asked how they become architects, one can answer that architects before Solution Architect are evolutionary along the technical branch, and corporate, either according to the technical branch, or according to the managerial branch, including business analytics. At the same time, you can become an architect at any number of years.

Solution Architect and microservices

The introduction of microservices begins with a business, when marketing starts doing experiments – requesting features in the form of MVP, then testing in the market, then either rejecting (which is rare) or refining. Improvement is required both after confirmation of the assumption, and erroneous in the form of an adjustment. For the operations department, this means rolling out a huge number of features that were developed in a hurry and can bring down the main application – the monolith. This service tries to run these changes in an isolated environment as a separate functionality, for which it asks the development department to develop them separately – in the form of microservices.

It is necessary to separate two levels of separation: technological and domain. Technological features in the work are the same, since that services, that its components, if it is divided into component parts, are technologically launched and supported in the same way. But, unlike services, which must be minimally interconnected with other services and must provide a specific API and SLA, the components are tightly coupled. The reason for the separation into components is of a technological nature. For example, an online store contains services such as the online storefront itself, payment services, warehouse services, and the online storefront is a service on the CMS Joomla! and a database management system (DBMS) MySQL. Here, the division of the service into its component parts occurred due to different software products written in different programming languages. At the same time, for the customer this is a single service on CMS Joomla! performing a single function of providing an interface for ordering to users online, provided by the hosting as a single service (it will not work separately), can work as a catalog of goods without other services (payment, ordering)… From a technological point of view, the components are services:

* Singles are not functional;

* Strongly connected, as many requests are generated for each request to the CMS;

* Interaction interfaces are complex and varied – not even the API is used here, but the SQL interaction language;

* Strongly connected functionally through a complex technical API – only known to the user supported compatibility of some CMS versions with other DBMS versions.

Dividing a system into microservices begins with an analysis of their boundaries, an analysis of the benefits of separation and the added complexity of a distributed system. It is better to separate microservices when there is a combination of need:

* Technological necessity, for example, a large load that is difficult to withstand without separation, for example, scaling, another type of support (SLA);

* For business, the dedicated service is already a separate and little dependent function – further we will consider the DDD (model-driven design + ubiquitous language) approach to the implementation of microservices;

* Requires change of technology platform.

Microservice meets the following characteristics, according to M. Fowler (martinfowler.com). They can be summarized as:

* 1. Must be a component or service. Each microservice is a complete, full-fledged independent service from the point of view of the developer, system administrator and user. It should be able to be easily replaced with another, both in the developer's code, both in the process of work (should be), and presented to another or removed in the user interface. Failure to fulfill the conditions of interchangeability at different levels lead to one service divided into parts – a distributed monolith;

* 2. Organization of business opportunities;

* 3. Products are not projects;

* 4. Smart endpoints and silly connections. Does not require complex integration with debugged services (the integration of complex systems is handled by a service oriented SOA architecture);

* 5. Decentralized management. This refers to orchestration like Kubernetes, network management like Istio, delivery management like KNative;

* 6. Decentralized data management. Due to the self-sufficiency of the service and independence from others, it must have an independent state – a database, and so that the choice of a database management system is independent – there is its own;

* 7. Automated infrastructure. The process of deployment, scaling and rollbacks should be automated, which allows you to quickly automatically rollback, fix the isolation of the service in the code;

* 8. Provided for refusal to work. To visualize failures, you can look at Jaeger and Prometheus, to localize problems, services must be isolated, represent one single

service, which allows you to isolate, limit the harmful effects on other services in case of failure and automate rollback;

* 9. Evolving design. The system grows outgrowths in the form of services – it becomes overgrown with them, while its structure does not need to be changed. Neal Ford and Rebeca Parsons in "Microservices as Evolving Architecture" focus on continuous improvement.

Business and business architect's perspective

Business architecture (Enterprise Architect) is the IT architecture of the entire company. It operates with abstractions and entities at the business level, these are strategies, business processes, services, and the like. The systems and interconnections that support the business are called the IT landscape because it contains many systems that do not form a single whole, connected by the business processes in which they participate and which are not limited by them. The Business Architect (Enterprose Architect) works at this level, adjusting the current landscape to the current needs of the business. Often, for traditional companies that did not develop in the high-tech sphere in the blue ocean, it is attracted when the IT landscape has taken shape and difficulties arise in its development and adaptation to changed conditions, a minimum capable product (MVP) is created in technology startups. The business architect of the corporate IT landscape must solve problems with a low rate of changes (due to the impossibility of local testing, postponing distributed changes, a breakdown claim after rolling distributed changes) – Time To Market, quick adaptations to user expectations – Customer Experience and cost – Cost… The first is tackled by the architect's sequential tidying activities, reducing cohesion and confusion, which simplifies and speeds up the process of making changes and, as in any knowledge-intensive field, where the main cost is the man-hours of workers, reduces the cost. More and more often, the architect is not provided with the requirements, he connects at the earliest stages of their formation, after his capabilities are severely limited. To do this, he needs to actively participate in negotiations and meetings in order to adjust the requirements. Further, to form several possible solutions with different levels of complexity, from simple and fast, but not effective ("solutions on the knee"), through the optimal, and to large-scale and flexible. Further, to form an architectural committee, at which to propose solutions for the choice and adjust the choice towards the optimal one.

Changing the existing architecture can be carried out in three ways: supporting the current one, completely replacing the current one, complementing the current one. Replacing the current one requires a long and lengthy study of the functionality of the current system, then finding out the functionality that is currently in demand and searching for differences between the current functionality and the expected one, after which the cost and development time are calculated. During the presentation, in most cases, a refusal will be received in the development of the system, since the customer does not need a technical update of the existing functionality, but he needs new ones and the correction of the old, and even more so not for the time and funds that were spent on creating the current system. With the consistent improvement of the system, fundamental changes to the system cannot be achieved, since the architecturally different functionality of one part contradicts others and changes in the architectural style are not achieved by successive improvements of the parts due to the complex nature.

The Enterpris Architect strategy implements different company strategies in different ways:

* Growth (scaling) strategy when the market is free – the architecture unifies and debugs processes;

* Strategy of innovation (search for hypotheses by Lean Startup). Creation and delivery of features and testing of hypotheses about the demand for these features are as fast as possible;

* Integration strategy. It is necessary to develop an open, most scalable and versioned API;

* Adaptation strategy. Not a strategy using Agile to adapt to the market;

* Strategy of quick decisions. Consistent changes.

Difference between Architects and Architects (NEW)

In general, the architecture of a fairly large company is called Enterprise Architect. It describes all levels of detail and the entire scope of the company. For any IT architect, IT interacts with the rest of the company's business. For most companies, the business makes money, and IT provides this opportunity, and understanding how this happens and what needs to be changed the task of the corporate architect. To do this, the corporate architect works with two layers that make up the corporate architecture: business architecture and IT architecture. It is very important to understand that in most cases IT is the backing function of the company, and the architect describes the current business architecture of the company and adjusts the IT architecture so that the described business architecture functions as intended by its creators. The creators of the business architecture are the heads of departments, functional blocks and divisions. They are the ones who know exactly how their departments should work. At the same time, they, too, often do not take it out of their heads, but rely on existing processes, for example, the sales department relies on customer experience that they have received in other companies and to which they are accustomed, or as a result of negotiations. The already formed processes in the business architecture take advantage of the power of IT, but this does not mean that the corporate architect needs to make the IT architecture a mirror image of the business architecture. On the contrary, if several processes have similar requirements, then the IT architecture should unify this, if possible, provide flexibility and scalability in a short time in the wake of changes and growth of the process, as well as offer more efficient ways to solve problems.

Political architecture (or layer of stakeholders) is a layer of stakeholders, their interests and agreements between them. Using the imperial method, it was found that for an architect it is 50 % important to collect information, another 30 % to negotiate, and only 20 % to a choice of technical solutions. With the availability of information, negotiations are easy and their result can be easily recorded. When looking for data, you need to find various solutions. This is a separate skill – the squeak of alternative solutions on the required data, called divergent thinking. During negotiations, it is important to listen to other people, offer different options, focusing on (sell) the best solutions, and find compromises. On the basis of already available data from a large number of solutions, it is necessary to find the best solution, which usually does not cause difficulties for quiet specialists, since they are accustomed to convergent thinking. For the architect, the main contractor (the main stakeholder) is the owner of the product for whom the architecture is being made. There is only one exception if decisions are made by other stakeholders, and the product owner simply broadcasts these decisions. Other stakeholders: the customer (or owned the product, as its representative), developers (minimization of work, clear setting of tasks, Time2Market), service departments (implementation of their standards), service architect (guaranteed service operation), corporate architect (landscape maintenance). If there is no common language, then either the architect of this area, or the owner of this area, need to escalate. For example, if there is no common language with the service team, you can escalate either to the service owner or to the service architect. If the language is not found with the owner of the service, then it is escalated to the architect of the service, if with the architect of the service, then the owner of this service. If there is no common language with either one or the other, you should ask yourself whose interests you are defending. The customer wants for the business:

* planned process;

* minimization of risks;

* movement towards the goal.

Business architecture is an enterprise and communication activity. According to TOGAF, it consists of pleased with the architecture, and since they are descriptive, they are often:

* organizational architecture (business goals, people, roles),

* product architecture (products, distribution channels, customer journey),

* process architecture, linking the organizational with the product, as the seller and the buyer processes.

The user path is called the client path. The path itself is the sequence in which services are used, which can be grouped into products. Service is what the client sees. To provide a service, a business needs to perform certain actions. Such actions that are repeated for different users are called processes, and the logic by which they are provided is business logic. Process is a business view of a relationship with a client. Processes are divided into business processes (for an external client), supporting processes (for an internal client) and control processes (for a company's work). All processes are placed in the process register and constitute the business landscape of the business layer. This layer also operates with other entities, such as needs, presence, and others. Processes themselves consist of a unified set of operations that programs can perform.

The architect investigates the ability to provide services (called business competencies), that is, the ability to provide the necessary operations with programs and describes them using competency maps. Graphical representations such as BPMN diagrams and structured textual descriptions such as process registers are used to describe processes and how they relate to the applications that implement them. From the processes, the user paths are built, along which the user walks, in turn, it is detailed to the level of the operation with reference to the application interfaces using the technological map, which is necessary for their development. Technological maps structurally detail the description of the process by technological objects and attributes. The process itself contains scripts, operations, roles, data, and the technological map adds API, functions, attributes to them. Thus, linking the operations of the process with the service programs that implement these operations. The routing does not describe the implementation on programs, nor the API – they are points for creating operations. Maps can be designed in various systems such as ARIS.

IT architecture is divided into:

* Application architecture – here are the systems and applications that are used in the business layer to implement their processes.

* Information architecture is a collection of information (data) that is exchanged between people or applications as part of the execution of processes. According to the level of logical abstraction, it is divided into: conceptual, logical and physical levels. It is managed (Data Governance) by artifacts, the main of which is the corporate data model. Data models are of different types, for example: relational, object-oriented, chronological (Time based), NoSQL and others.

* Integration architecture connects various components of the system and it is these connections that describe the integration architecture.

* The technical architecture describes the implementation of the previous layers. It itself is also divided into layers, but these layers are not abstractions, but are a technical implementation in relation to the client. These layers are also called a stack, since they are not located from the user of the business layer one by one in a specific order and not a single layer can be skipped, while the layers of architecture describe the architecture in a different way, can be supplemented and changed, but we adhere to the set of layers accepted in TOGAF. So, the stack of WEB-applications consists of:

** Application layer – the layer with which users directly interact and which provide processes and is implemented using WEB-interfaces in browsers. Execution of business logic at the level of WEB-interfaces is unacceptable – all work is delegated to the underlying layer.

** The network layer ensures the operation of WEB-interfaces in the user's browsers, transmitting data. The upper layer of applications is needed by users who are comfortable using the graphical interface and so that any routine actions are performed for them. The network layer provides communication for server applications such as WEB servers and DBMS.

** The hardware layer is represented by runners. These devices can be hardware-based with varying degrees of versatility. For example, a load balancer can be purely hardware, it can be hardware with changeable firmware, it can be software executable on a general purpose computer of the x86 type, it can be launched both directly and in a virtual machine, and in a container – all these are implementation details.

** The storage layer is made up of storage devices. These devices can be specialized devices such as IBM DataPower or regular RAID with a control module. The data in it completely describes the state and is the result of work, and the previous layers are only needed to change and provide convenient access to users.

If necessary, other layers can be implemented, for example:

* information security layer implemented by the firewall;

* a layer of basic containers;

* layer of local fault tolerance (HA) using the example of the Kubernetes layer;

* containerization layer;

* virtualization layer;

* a layer of resilience to failures implemented by load balancers on different DataCenter.

In any case, the number of layers is standardized, those that differ are indicated, so that each layer belongs to a specific info-structural department and an operation department.

Let's pay more attention to the integration architecture, since this is the most critical layer for the architecture. In this layer, connections can be presented both in a graphical form (in the arrows on the diagrams between systems), and in a tabular form – in the form of a description of the supplier, consumer and contract (the supplier's obligations to the contractor). The arrows point from the supplier to the consumer, that is, in the direction of the integration flow, while the service modules are not indicated. Depending on whether the parameters are functional or non-functional, they will be described either by API or SLA. Also, depending on whether the connections are inside or from outside to inside and outside. The first type is more visual, and the second allows you to give more detailed characteristics.

The system itself can be integrated in different ways, such as:

* direct integration (communication via API point-to-point, advantages: minimum overhead, disadvantages: two-way revision of systems is required, complexity of change management, complexity of scaling, no reuse);

* using gateways (communication through the API of an integration layer, such as a queue with a firewall, advantages: minimum overhead, unified API, disadvantages: complexity of change management, complexity of scaling, no reuse);

* Enterprise data bus or enterprise service bus (ESB) provides asynchronous umbrella integration based on the principles of event and service approach (SOA, service-oriented architecture). The corporate data bus is able to flexibly route messages from one service to another. (advantages: unification, reusability due to SOA, replaceability of services due to SOA, disadvantages: an expensive solution in many applications, delivery time from tens of milliseconds);

* Service Mesh, like ESB, is umbrella, but applications do not need to integrate with it, since applications running in a containerized environment immediately receive integration. (microservices, advantage: minimum overhead, not noticeable for application developers);

* Integration file gateways and point-to-point file transfer (file overload). Point-to-point file transfer is the same point-to-point transfer, but it allows you to transfer large data in exchange for the transmission speed (advantage: it is possible to transfer very large

amounts of data, high delivery guarantee, weak connectivity of integrated systems, greater control, broadcast mode, disadvantages: transmission speed, possibility of desynchronization, high security requirements). Communication protocols are CIFS (Common Internet File System), NFS (Network File System extends the local file system) and S3 (Simple Storage Service provides access to object storage such as Minio and Ceph) and transfer protocols HTTPS (HTTP + SSL), SFTP (SSH + SSL) and FTPS (FTP + SSL). From the point of view, records can be divided into block (disk) and object (writing to the key-value database: Bucket).

Systems can communicate in various ways:

* integration request (normal synchronous request-response),

* remote procedure call (RPC),

* sending a command to the queue (from the supplier to the consumer directly through the event queue),

* publish-subscribe, Push-Sub (sending events to a common queue, from which groups of events are retrieved from the system in advance, undefined by the provider),

* packet data transmission,

* transferring files to storage,

* streaming data.

The interactions themselves should be described, and preferably unified. To describe functional or non-functional parameters (response time, availability, message size, bandwidth) of interactions between the supplier and the consumer, a Contract is used, which the supplier undertakes to fulfill. Functional parameters are described using the Application Programming Interface (API). Service APIs can be divided into groups based on message format (DTO, JSON, XML, binary) or protocol (HTTP, REST, SOAP). It is important to specify in the API contract: ID, name, version, purpose, template, specification of input and output parameters. The API itself will contain methods (encouraging the consumer to take data, change, allocate, etc.). Parameters passed in a method are described by a method specification, for example, using OpenAPI or

Swagger. For many languages, and primarily for the Java language, you can automatically generate a specification for OpenAPI using Swagger by Javadoc annotations (by special comments) in the code. The specification will be displayed in both text formats (JSON and YAML) and graphical. For ease of design, you can use the Swagger Editor (https://swagger.io/tools/swagger-editor/). This helps to organize automated contract testing. In general, the presence of the specification helps to organize API-first development, when the API specification is first written, and already the application code is developed for it, implementing it.

To bind an API to the system providing it, the endpoint concept is used. Endpoint is described either by an address with a port, or MMT, hiding them. APIs are described by a machine-readable description (declaration), for example Java interface and WSDL document.

The API life cycle is close in stages to the software life cycle and the following cycle stages can be outlined: requirements gathering, interface design, implementation, testing, operation, and decommissioning stages: decommissioning warning (marked as obsolete to avoid new users), decommissioning (tracking the decrease of existing users), disposal (closing access to the API and removing it), and the stage of creating a new version. The new version of the API can be compatible and incompatible with the old one, for this you can differentiate them by semantic versioning into incompatible with the previous version, compatible with the previous version and bug fixes. By semantic versioning, the version can be written as "major.minor.patch", for example 1.9.0 – > 1.10.0 – > 1.11.0.

The solution integration layer is described in the conceptual architecture. The conceptual framework only indicates integration. It is intended for coordination with stakeholders: architects of the integrated solution, corporate architect responsible for the landscape, product owners, owner of the variable landscape, security service, resource provision service (infrastructure department) for its quick approval. The content is indicated in the target architecture (detailed architecture) of the solution. She participates in the early stages of creating a service and in the acceptance tests of the solution within the framework of architectural control. The conceptual architecture is the source of artifacts for implementing and developing a detailed solution architecture. We can distinguish the following stages of creating a conceptual architecture: planning, creation, development of an incremental draft (sketch), approval, revision. In order for the diagrams to be understood by both the developer and the owner of the product, a short description (Architecture Decision Records, ADR) is needed: the goal, the essence of the solution, rejected options, rationale, consequences, affected systems, a link to the diagram and the originator's contacts.

The corporate data bus is usually used to connect individual systems. For connections within the system – point-to-point network connection and their control using integration gateways. Integration gateways can and can be used between systems, but often, these systems are not protocol compatible and require their transformation, which is provided by the corporate data bus, but not provided by the integration gateway. And the use of a corporate data bus to connect the components of one system is expensive, since the asynchronous protocol will be loaded by other systems due to the fact that the message queue that implements it will be filled with messages from other systems and subsequent messages will wait for their sending, which causes delays in data transmission… For the exchange of large amounts of data between systems and within systems – distributed information storage.

With the layers we figured out, now, imagine the architecture as a layer cake. We can cut off a piece from it and narrow it down in more detail, but the number of layers in all pieces of this pie will be the same. By analogy, we can divide the corporate architecture into chunks. These pieces can be separate systems that are directly used on the business layer, or it can be a group of these applications developed by some department or department. The pie itself is the corporate architecture, and the piece is the solution (service) architecture. Enterprise architecture appears when we need to add new pieces to the pie, while the pieces are the architecture of the solution, and the rules according to which are made, for example, the number of layers – corporate architecture standards, cakes and cream – providing technologies for corporate architecture.

Enterprise Architecture

Corporate architecture is the architecture of the entire IT landscape of a company. A corporate architect is guided by the principles of creating architecture, a kind of constitution. The principles are described in the third section of TOGAF 9.2 under clause 20. They govern which requirements, for example, the principle of customer focus or lean manufacturing will meet. It is important that all participants (stakeholders) agree to adhere to the same principles. The principles are categorized according to their applicability to the architecture layers: business, data, application, and technology.

In practice, corporate architecture develops in three directions: unification of technologies, development of conceptual architectures, unification of the landscape. A landscape is understood not so much as an AS map, but as a set of unified solutions on the basis of which a business system is built, and with which it interacts, usually with infrastructure systems (logging, monitoring). To build the business system itself, unified solutions and technologies are used. To unify solutions, old solutions are adapted or new solutions are created, the customer of which is the department of corporate architects. To unify technologies, working groups of corporate architects are created – researchers who test the capabilities of existing solutions, analogs and make decisions either on the distribution of existing ones, or on the implementation of new ones. On the basis of research, architects – researchers create standards that describe the boundary possibilities of their applicability and regulate their use. According to GOST 1.1-2002. Standard: "a normative document that is developed on the basis of consensus, adopted by a recognized body at the appropriate level and establishes rules, general principles and characteristics for general and repeated use concerning various activities or their results, and which is aimed at achieving an optimal degree of harmonization in a certain area". It is important to emphasize that the standard fixes the agreements already found and that the stakeholders are interested in its implementation. If this is not the case, then the standard will not be met. Depending on the activities, the corporate architect may have different customers: the management of the organization, the regulatory departments (security, support, and others), the development teams of platform solutions, the architects of the development teams (to create a conceptual architecture of their future service). To maintain standards compliance, their requirements need to be validated through automated means, such as embedded in development tools and DevOps, or running validations at runtime. For this, it is necessary that the requirements of the standard are not only formalized, but also machine-readable. Not only the implementation, but also the architecture itself can be checked automatically. At the same time, checking the architecture should be carried out especially carefully, since its change over time is especially expensive afterwards. The principles of checking the corporate architecture and the solution architecture are different, since the corporate architecture is not just the sum of the designs of all solutions – it has different tasks.

When building a business system, it will be opened from existing unified solutions, new functionality, and already existing technologies and infrastructure systems will be used. For example, unification and standardization of a set of programming languages. Here, the criteria are, among other things, the economic indicators of the development cost (the speed of development in a certain language and the cost of the developers themselves with the necessary qualifications in this language), guarantees for support (the availability of a sufficient number of free personnel and resume), the complexity of support, outsourcing and others.

Enterprose Architect participates in the service development process at least in two stages – checking the developed conceptual architecture of the service and checking the compliance of the detailed architecture of the developed service with it and the standards for acceptance tests. In practice, he makes the conceptual architecture of the service himself and adds it to the service map himself, so he has the necessary experience and knowledge of all standards. In practice, Enterprose Architect assists the service architect in developing the detailed architecture of the service in its drafting and conceptual architecture in accordance with the vision of the service architect. In fact, the conceptual architecture is the architecture for integrating a service with other services to bring it into the service landscape, while the detailed architecture is the implementation of the service in accordance with the expectations of stakeholders (customers, controlling departments). It is the implementation that must comply with the restrictions imposed by Enterprise Architect on the implementation, for example, to unify technologies. If there is no collaboration, then Enterprise Architect becomes a regulatory body that blocks the deployment of the service with critical remarks or sets a technical debt with a deadline for elimination.

An enterprise architect, when developing a conceptual architecture, negotiates with the service architects with whom his service needs to integrate and finds compromise solutions, as well as other stakeholders, such as security personnel. Having agreed, he brings to a higher level of agreement, to confirm the consensus. An enterprise architect needs knowledge of services and standards, communication skills and knowledge of integration architect, such as an enterprise service bus and the ability to quickly create an architecture with insufficient information in a tightly constrained time frame.

A more lenient auditing-based standards review process can be applied, which can often be boiled down to checklists or automatic compliance checks. In practice, the service architect can go through such checklists under the guidance of Enterprise Architect for correct interpretation at the very beginning and then go through as necessary to correct compliance. Modern systems provide their setting in the form of configuration files, which allows automating the check against a list of such systems and their configuration files. Most of them work in asynchronous mode, that is, the states of the corresponding systems are asynchronously brought to the state described in these configuration files – IaaC. In most cases, the state to which the system is reduced can be obtained in a declarative form in either JSON or YML format. These formats are compatible and mutually convertible. There is an OPA (Open Policy

Agent) project by the CNCF (Cloud Native Computing Foundation) consortium that allows you to create rules in a declarative form to check that they match JSON configurations. The check is carried out in a synchronous and asynchronous manner. For most systems, the application of changes is possible in the form of IaaC, that is, as sending new configuration files to the systems, on the basis of which the state of the systems is brought into conformity with them. In fact, the differences between them apply. This means that you do not need to check the current state of the system, but you can check the poisoned configuration files themselves. As a result, we have a configuration file for validating changes in a declarative format – GaaS (governance as a code). Let's look at a few layers to check:

* Layer of virtual machines (OpenStack, VMWare vSphere JSON Template);

* Network layer (VMWare NSX);

* Server configuration layer (Ansible AWX);

* Service configuration layer (Hashicorp Terraform / AWS CloudFormation);

* Layer for configuring service containers (Kubernetes / OpenShift);

* Layer of traffic routing between services (Istio, Envoy);

* Application libraries layer (NPM for JavaScript, Maven for Java, Composer for PHP).

The interaction between services (traffic routing) is described by the Istio and Envoy configuration files (more fine-tuning), which are submitted to Kubernetes and are Kubernetes configuration files. OpenShift provides a Kubernetes extension, but its config files are Kubernetes native too. Kubernetes itself is configured using YML or JSON configs transmitted asynchronously. For example, Kubernetes configuration files fully describe the state of containers (kubectl get deployment – o yaml), allowed inbound and outbound traffic from the service (kubectl get NetworkPolicy – o YAML), service accounts (kubectl get ca – o yaml), encryption between services when applied Istio (kubectl get Destination Rule – o yaml) and so on.

Many cloud providers that provide APIs for their service management clouds either have their own IaaC configurations on top of them, such as AWS CloudFormation, or integrate with the Terraform abstraction for which you can develop your model. The configurations themselves are described in a declarative form, but in their own configuration language. But, you can get the state of the reduced system state in JSON format:

```
terraform init
```

```
terraform plan – out tfplan.binary
```

```
terraform show – json tfplan.binary> tfplan.json
```

Ansible AWX is built on top of Ansible and accepts its configuration files. The configuration files themselves are written in YAML format. Ansible converts the state of servers from the list (in the inventory file) to the one described in the configuration file. Allocation of servers in OpenStack, to some extent, can also be described in YML format. At the level in VMWare NSX describes intersegment communication in configuration files in the same YML format as others. If we talk about the library layer, then many builders install and install packages according to the configuration files, so NPM in NodeJS works with JSON configuration package.JSON, Composer in PHP also works with JSON configuration composer.JSON. Conda in Python uses the conda.YAML configuration file in YAML format, which is unambiguously converted to JSON. The exception is Maven in Java, which stores XML configuration in the pom.xml file, but, as practice shows, it is not difficult to convert pom.xml to valid JSON format using Python / NodeJS.

Solution architect

The solution architect (Solution architect, Software architect), for example, a service or system, is responsible for the detailed design of the architecture of the developed solution and its API. As part of the solution, he defines the detailed design of the solution, manages the dependencies and technical debt of the solution. His work depends on the enterprise architecture (standards), the architecture of the area in which his solution belongs, and the architecture of the platform he uses. His work is judged by:

* quality and speed of development of a detailed architecture of the service,

* reuse of developed components,

* dynamics of closing technical debt.

It connects at the earliest stages of service approval by management, so that it can influence decisions regarding this service before they are made. Moreover, these are not technical decisions, but managerial ones, for example, timeframes and budget. If these decisions are made without it, then the architect will not have to decide the choice of the stack and the type of architecture, but choose what and to what extent he can implement. The architect also oversees the implementation of architectural solutions and architectural control when accepting the service. Usually, the service architect accompanies the service itself and at the reception, protects the architecture and carries out the acceptance of the work before the customer at the acceptance tests.

A service architect is a role, while by position he can be a developer, a database engineer, and a business analyst and director. Usually, when a direction is being formed, this role is played by the forming director. Later, when individual services are known, he connects either technical specialists with high communication skills, or a business analyst with an assistant in the form of leading technical specialists of this service.

Most often, this role is assumed by a technical specialist, since it is practically impossible for non-technical specialists to build up a technical base so that they can descend from the upper layers of abstraction to the lowest and see technical problems and bottlenecks. It is the ability to switch between different levels of abstraction and work effectively at them that is the key skill of an architect. But technicians use their communication skills to attract technicians where they lack expertise, which technicians are not used to doing. This is going away because the technicians do not have enough resources to cover everything themselves. Out of habit, when they had one task in their professional area 1, they plunged into it to solve it, in the role of an architect, they cannot become specialists of all technical profiles and do everything themselves, there are not so many resources for this and from them this and not required – the team and the specialists involved in the project will implement it. The downside is that technical specialists, by virtue of the ability to delve into the problems of each component, can take this into account, but this is possible on small projects, on larger projects it is solved by involving experts and corporate architects practitioners for auditing, following standards, which is familiar to managers… Another difference is the understanding of business requirements, in particular in the financial area, timing and integrations. If technical specialists, as a rule, do not have problems with integrations, then non-technical ones, on the contrary, shifting them to others, they may find themselves with the fact that it is difficult to put everything together. On the other hand, if you have strict financial and timing requirements in custom development, the project can fail, since the customer can refuse it and still demand a forfeit, and due to weak communication skills – at the end of the project. It is also observed when an architect defends a project from directors, especially from a financial one, when the architect cannot justify the choice of the technology he likes, but expensive, when in this situation there are no obvious objective grounds, for example, Java instead of PHP, Oracle instead of MySQL, micro-services instead of a monolith, a self-written solution instead of a CMS for a small online store.

In general, the tasks of architecture development are the coordination of components, the choice of new technologies, the achievement of the desired result without alterations, and the simplification of acceptance. The main task for managers of architecture development is to accelerate implementation and accelerate the entry of new employees. For the customer – compliance of the implementation with the intention and in case of deviations during the development of the correction at the early stages, which reduces the cost of fixes. For developers – quick and easy implementation into the production process.

Ideally, architecture is immutable, but in reality this is often not the case. Basically, the cornerstone is the communication skills of the architect, who knows how to negotiate, find compromises and communicate solutions. To convey the essence of the architecture being developed, various mappings, slices are used, which display the architecture from different sides. For IT, this is the development of the architecture of various layers. The layers can be by TOGAF: business architecture, information architecture, Solution Architect, integration architecture, technical architecture). At each level, it is necessary to display system components (structural diagram) and business processes (dynamic diagram).

In general, architects can be divided into two groups: Enterprise Architect and Solution Architect. Enterprise Architect is concerned with finding and unifying technologies, while Solution Architect is developing the architecture of a specific system based on approved technologies and entering it into an application map. In small companies, in which the developed system architectures are small, the corporate architecture does not stand out – it is replaced by a component of the system architecture, namely the integration architecture.

Solution Architect must have very good soft skills (communication skills). In everyday life, the image of a person sitting and drawing squares and arrows between them can be formed. But let's imagine a situation: an architect comes to a project, sees a team developing something and hears words from a customer: the product slows down and works unstable, the situation needs to be corrected. What, where and why it slows down and crashes, and just where his project is not clear. No deep technical skills are needed now, and projects are different (an architect is not needed on a standard project) and there are already experts on it. Here the main difference is not in the level of work, in comparison with the developer, but in a fundamentally different – instead of solving the task, the formation of the task itself. If a developer takes a ready-made task from a task setting system (for example, Jira) and performs it, sometimes clarifying the conditions with the director, then the architect spends a lot on finding stakeholders, developing an optimal solution based on requirements and finding a compromise solution. If you contact the manager, team lead, developer, then none of them will tell you what the problem is. If you go to the infrastructure, then the situation is even more interesting – someone, something, somewhere deployed, but somehow it all works, and who to ask the question is also not clear. If the system is more or less complex, then there are a lot of integrations, and behind each there are separate people, who can often be found by learning only from those who happened to work with them. At the same time, it will not be possible to directly ask the integrators a question – they have their own tasks, and often they do not get in touch. This is a typical situation of a manager in a project, therefore, in this part, the software architect can be viewed as a technical manager who manages not tasks, but architectural components.

Let's take a closer look at the communication and interactions of the service architect:

* He is in contact with TeamLead to view the current team backlog. The backlog contains a roadmap and requirements. This will allow and determine the amount of work that she can do and their priorities.

* He contacts the Product Owner to obtain a business description of the required service, and on what features he agreed with the customer, business processes.

* He liaises with the corporate architect to get the conceptual architecture of the functional area of the service and the requirements of the standards that the service falls under.

* He contacts the platform architect to get the component base.

* He contacts the server architects with whom he integrates.

* He contacts with specialists responsible for procurement (if new servers, paid software are used), security, technical support of the service (if new stacks are used), support, legal department (if OpenSource products are used).

When joining a project, like a manager, it is necessary to familiarize yourself with what is available and identify the stakeholders (stakeholder). Stakeholder – related to the project. He may be interested in success or, conversely, not interested. ITIL4 (IT Infrastructure Library version 4), a collection of recommendations for the use of ITSM (IT Service Management) from 2019 created by more than 150 authors to build IT developing since 1986, defines such basic types as Sponsor, Customer, User, Service Provider and others:

* User – a person who uses the service. For example, if the system is developed for the accounting department, then the accountants will be the users, although the customer can be the head of this department. Often they will test the product being developed and evaluate it;

* Customer – make demands and is responsible for the result from its consumption. For example, if the system is being developed for an internal department, then the head of the department orders this system for his employees in the expectation that their work will become more efficient;

* The sponsor is the resource owner who approves the payment. For our example, it may be a CFO who considered it economically justified to develop this system to increase efficiency for such a volume with such risks in the current situation;

* Contractor – an interested party responsible for the provision of services. The contractor can be both internal and external. For architecture, they can be a development team, system engineers, individual departments integrating with their various services, and others;