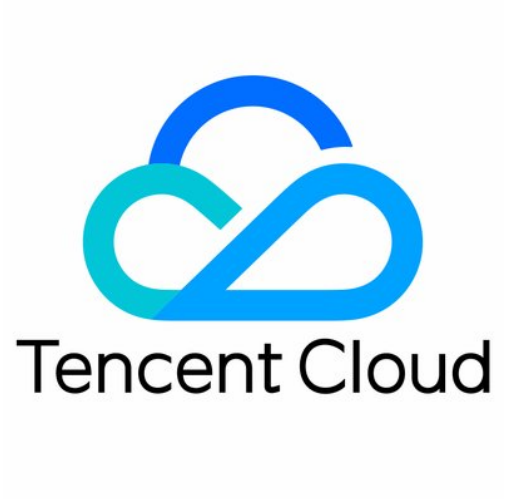
Serverless Cloud Function

Success Stories

Product Documentation



Contents

Success Stories
Tencent Online Education

Last updated：2024-12-02 16:29:12

This document shows the success story of the use of SCF in online education made available by IMWeb, which boasts the following achievements:
Affiliated to Tencent, the IMWeb team is a leading professional frontend team in China.
Focusing on the frontend field for many years, it has been taking charge of businesses with hundreds of millions of daily access requests such as QQ profiles, QQ sign-up, and QQ groups.
Currently, it focuses on the online education field and is dedicated to developing three major products: Tencent Class, Tencent Penguin Tutoring and ABCmouse.

Technical Attempts

IMWeb has made various technological attempts in the traditional web application area, such as traditional offline package and PWA offline application. However, each technology stack has its own strengths and weaknesses. The multi-dimensional comparison between the team's current technical schemes is as shown below:

Scheme Name	Above-the-fold Time	Development Difficulty	Maintenance Costs	Iteration Costs	Network-Dependent	Client-Dependent	User Experience	SE Friendl
Async rendering	1.2s ±	Low	Low	Low	Yes	No	Poor	Aver
Offline package (async rendering)	900ms±	Low	Low	Low	No	Yes	Good	N/
SSR	700ms±	Medium-high	Medium-high	Medium-high	Yes	No	Average	Goo

The comparison unveils that server-side rendering (SSR) has outstanding performance with regard to above-the-fold rendering and SEO. Based on the result, the team favors the SSR scheme for the following reasons:

SSR application performance

The following performance optimization issues are highlighted:

Processing of a large amount of CPU-intensive computation: SSR used in React-like applications works by calling the `renderToString` method of React on the server to render the React component as HTML strings. For complicated SSR applications, this process may involve a large amount of CPU-intensive computation, which is not a field of expertise of Node.

Improvement in above-the-fold rendering performance: As offline packages depend on the environment (application), a complete solution for caching relevant pages is required in a traditional web environment in order to improve the above-the-fold rendering performance.

SSR Ops costs

Service availability: As most frontend engineers may not be good at service Ops, service availability is also an important concern in technical scheme selection.

Based on the issues mentioned above, the considerations can be summarized as follows:

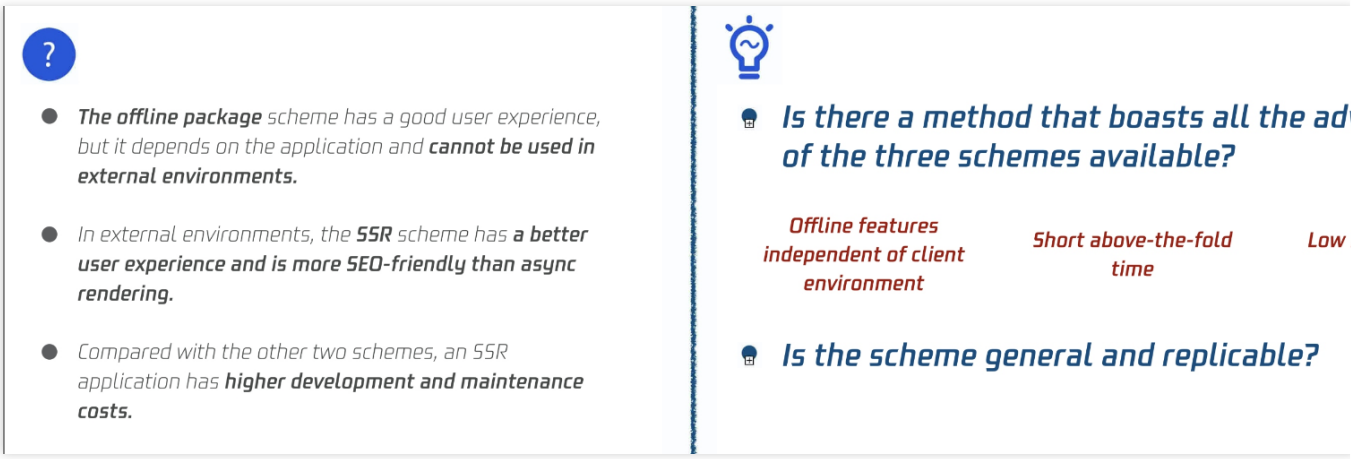
How to design a method that boasts all the strengths of the three schemes available?

Offline features independent of client environment.

Short above-the-fold time and great SEO experience.

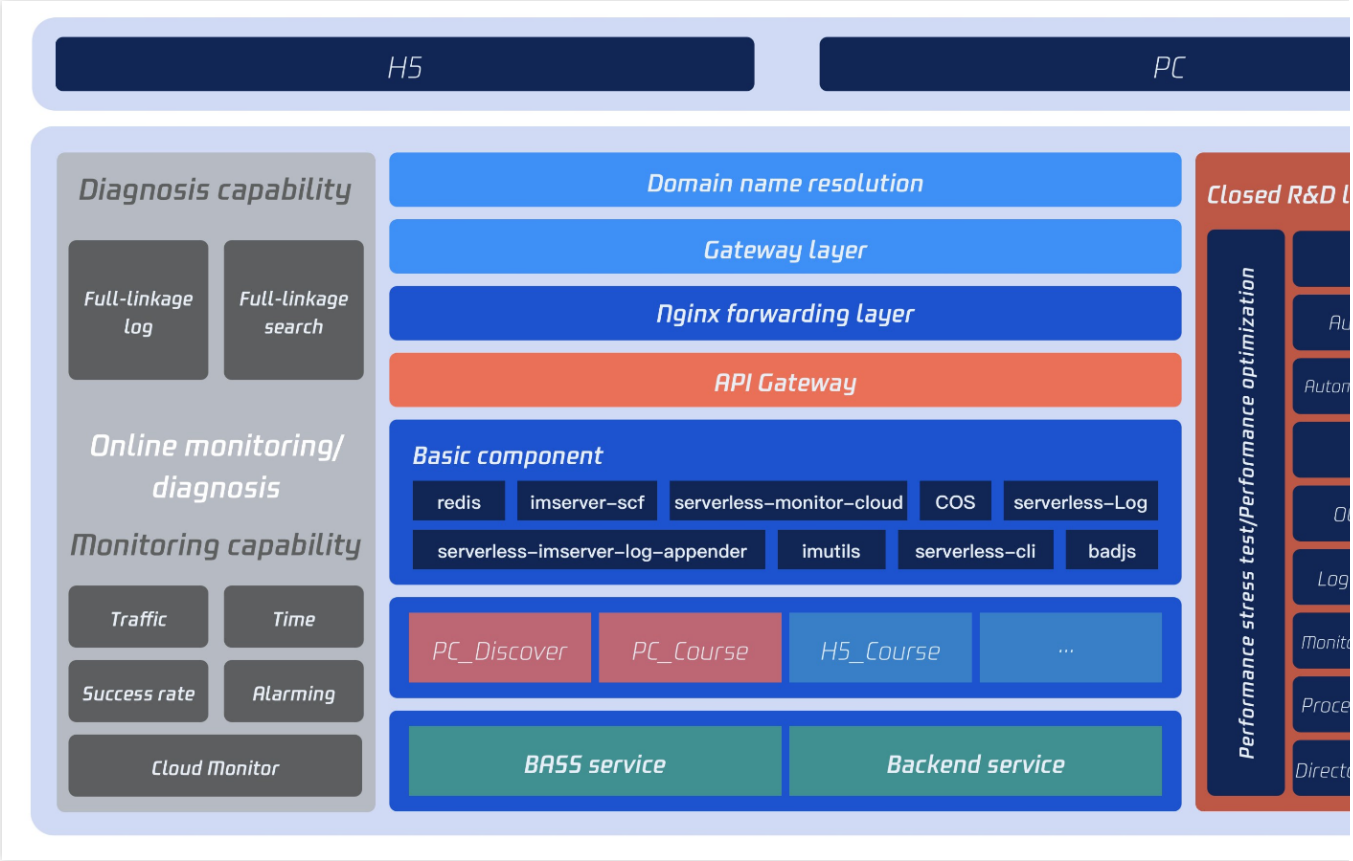
Low Ops cost and easier troubleshooting

Is the scheme general and replicable?



Overview of SSR Architecture Scheme by IMWeb

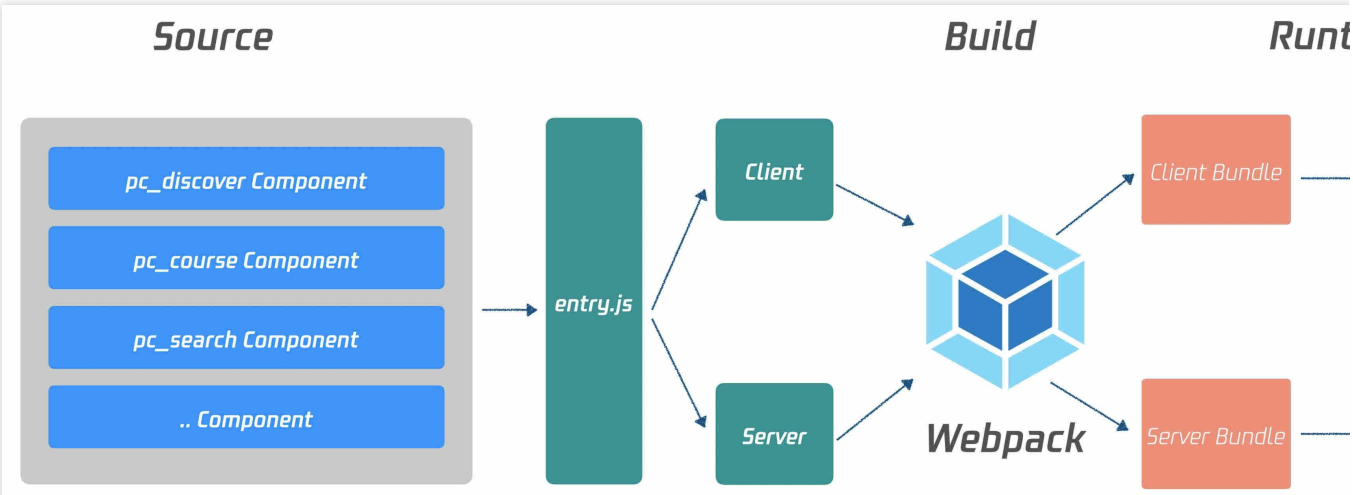
The SSR technical architecture used by the team is as shown below:



The following sections describe the team's existing schemes in terms of code organization, performance optimization, and runtime context.

Code organization

A heterogeneous mode is used in both the PC and HTML5 projects to construct SSR applications as shown below:



In heterogeneous mode, business developers can focus more on the business features without being distracted by runtime problems; however, they also need to pay attention to the following:

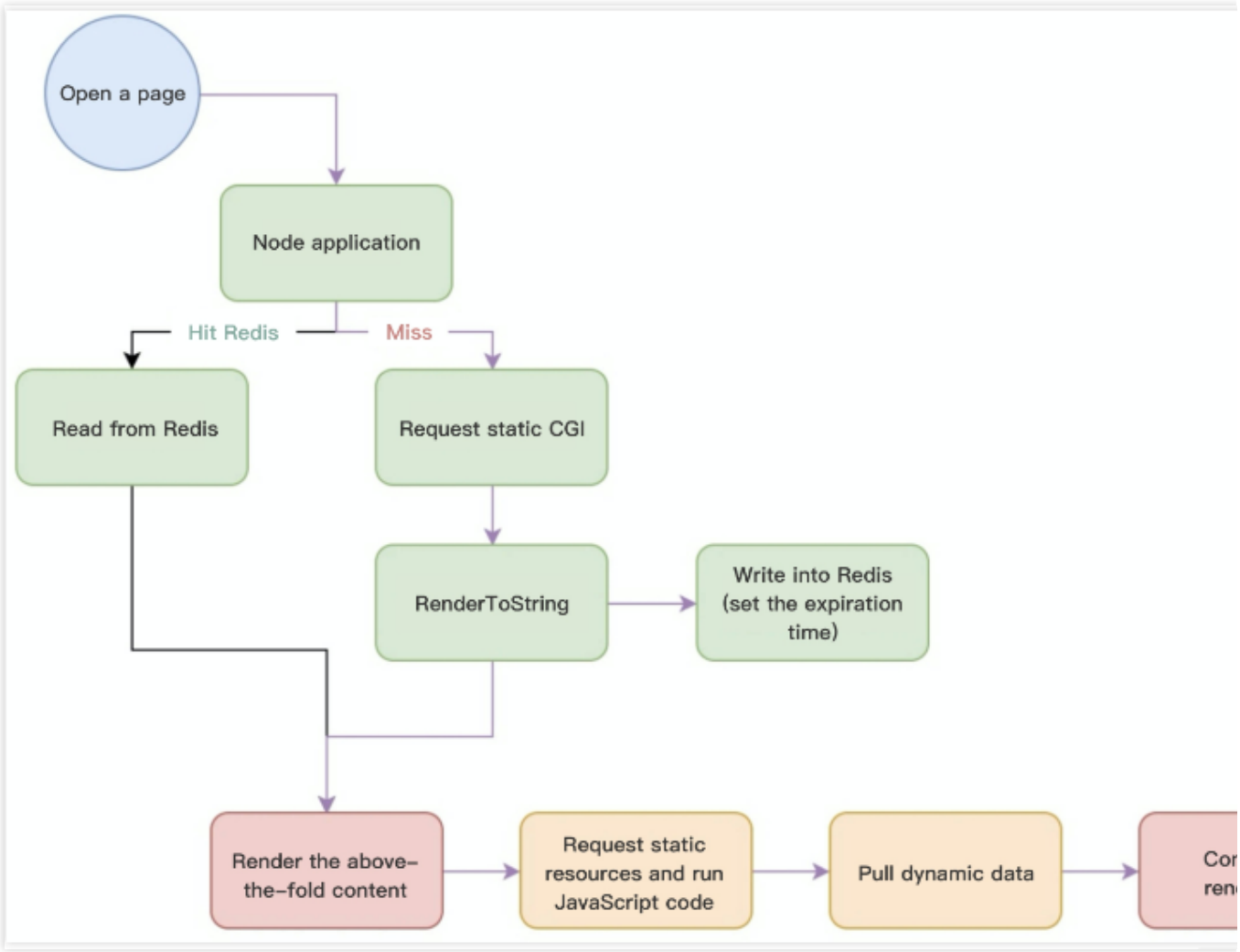
Correctly use constants such as `window` and `document` in traditional browsers.

Ensure that the HTTP data request library can support both the server and client.
Reasonably use the lifecycle of React applications.
Inject environment variables to differentiate between current runtime environments.

Performance optimization

Separating dynamic/static API data

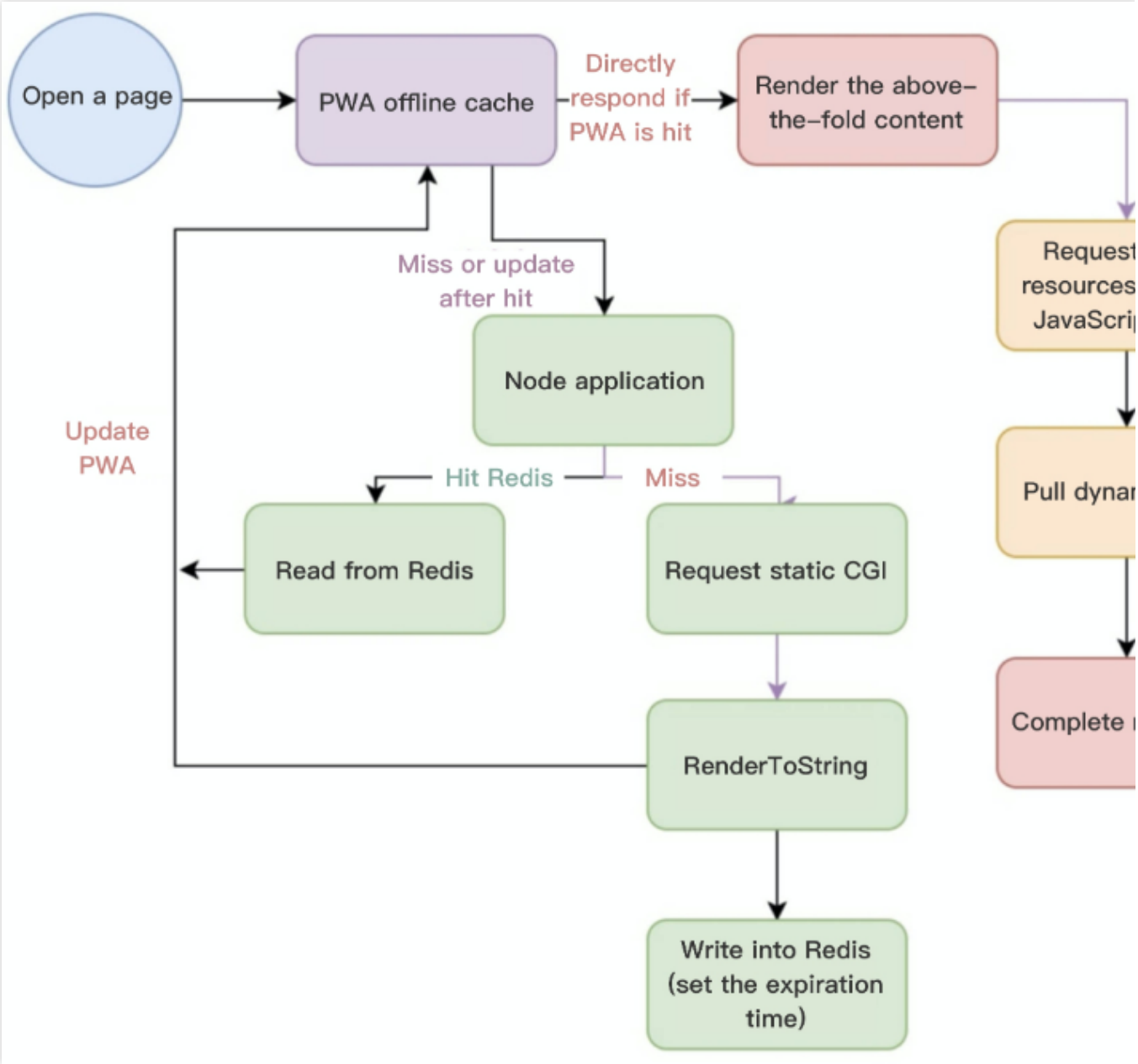
Page rendering usually depends on backend data, which can be divided into dynamic data and static data:
Static data: Data that rarely changes in a page, such as course title and description in Penguin Tutoring.
Dynamic data: Data related to user login status in a page, such as whether a course has been purchased and current course discount.
By separating dynamic/static data for APIs, the static data can be used for caching thanks to its low latency sensitivity. After static data is used to render the page on the server, the dynamic data will be used to perform secondary rendering on the server. The main logic is as shown below:



Redis is used to cache the page rendered with static data, which not only accelerates SSR rendering but also improves the QPS of a single server (`renderToString` is a CPU-intensive operation in a certain sense).

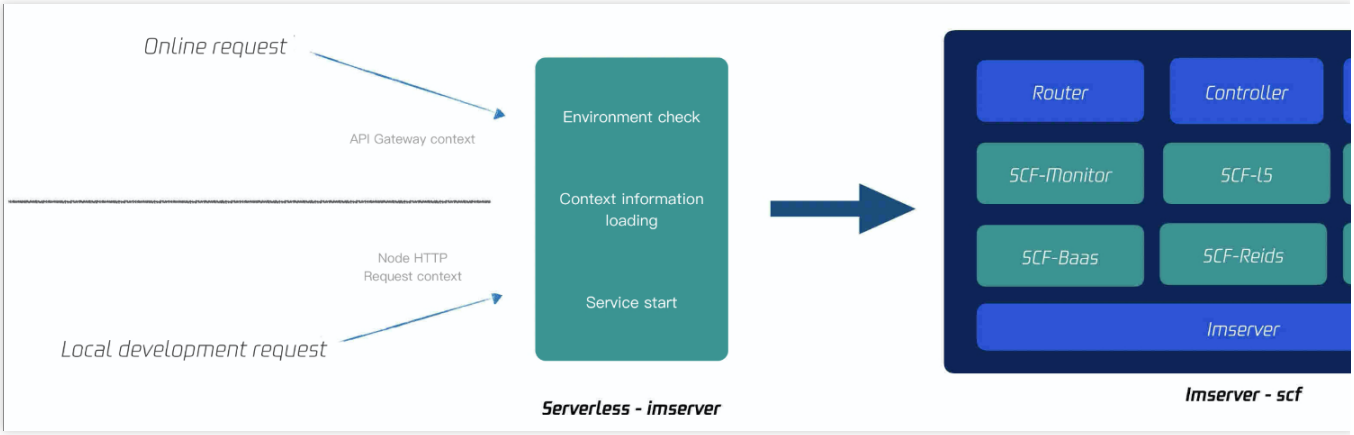
Using PWA in browser for offline caching

On the client, PWA can be used to cache static data-based HTML SSR pages offline so as to further improve the above-the-fold rendering performance. The main logic is as shown below:

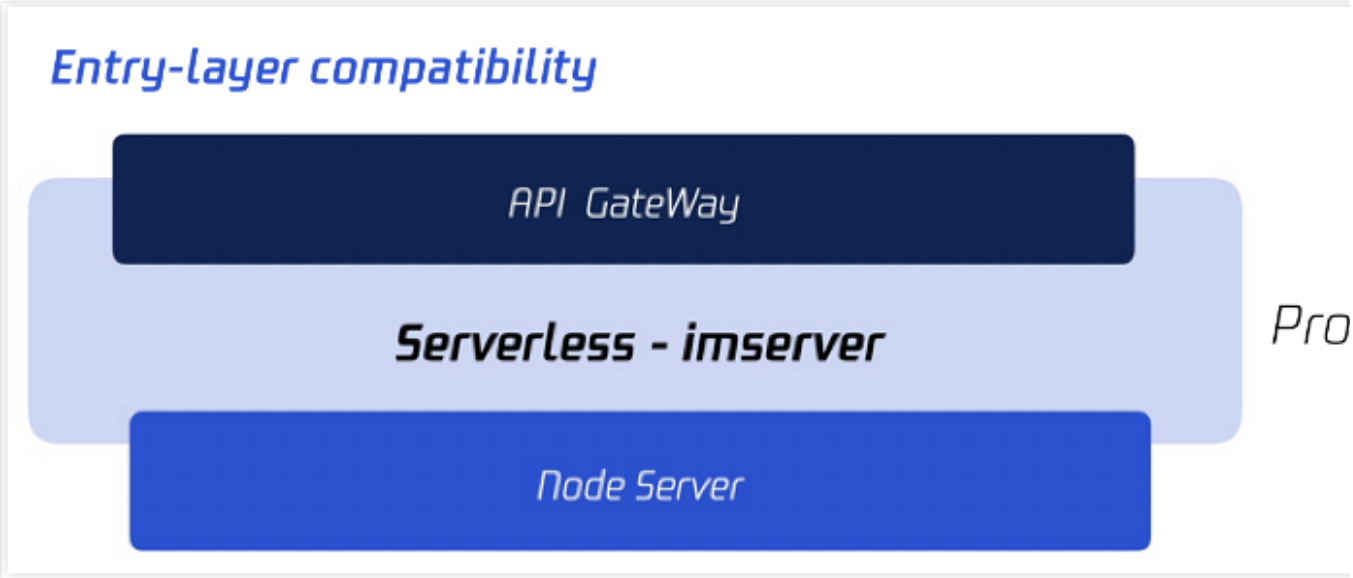


Runtime context

Because of the high Ops complexity and maintenance costs of backend applications, Serverless (Tencent Cloud SCF) is used to deploy SSR applications. Thanks to the inherent advantages of the Serverless architecture mode, the team does not need to concern over issues such as service Ops and scaling.



As shown above, in essence, an SSR application is a Node application, and SCF invocation is an event. Therefore, the following method is used to ensure compatibility with the two modes:
Based on the various standardized APIs provided by Tencent Cloud Serverless Cloud Framework, Serverless-based encapsulation is added to Tencent Cloud's proprietary Node framework (imserver). In addition, compatibility with the elements from `Event` to `Koa Request Context` has been added to the entry as shown below:



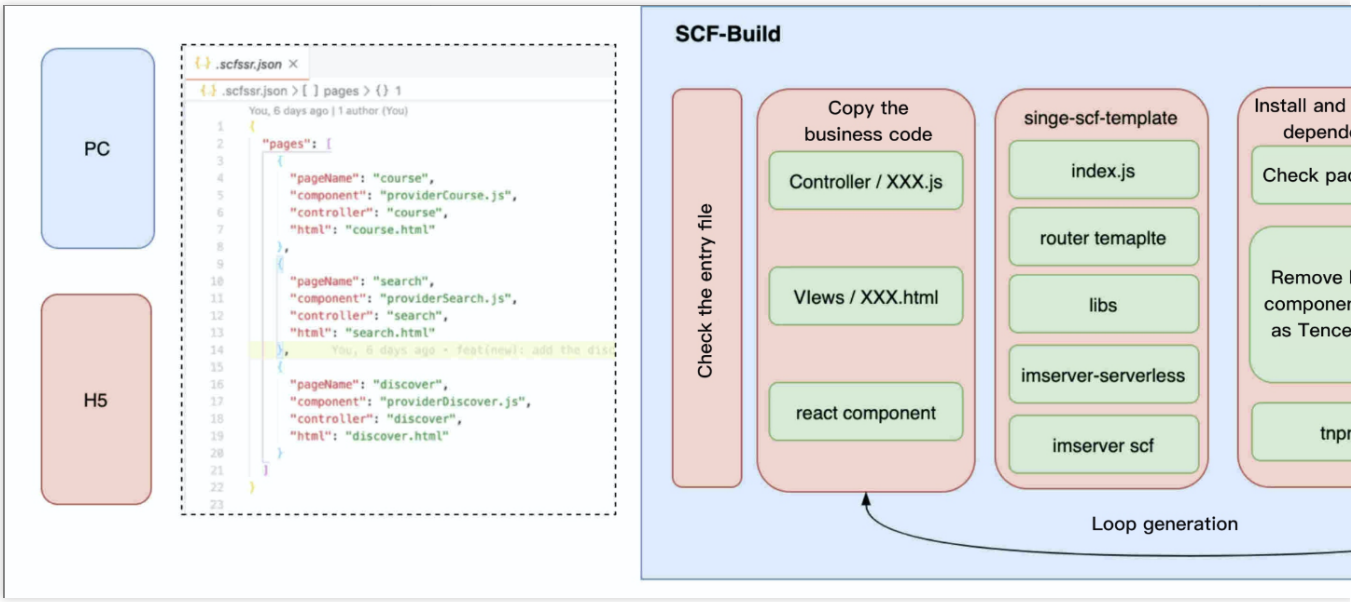
Problems in SSR Technical Scheme Implementation

Problem 1. SCF function division

As a business has multiple pages implemented through SSR, if SCF is used to implement SSR, it is required to determine whether the business features should be combined to one SCF function (at the business level) or divided into multiple functions (at the page level). The latter is recommended for the following advantages:
SCF functions are independent from each other. If the function of page A fails to be invoked, the function of business B will not be affected.
The function package size can be reduced. A smaller package size also shortens the function's cold start.

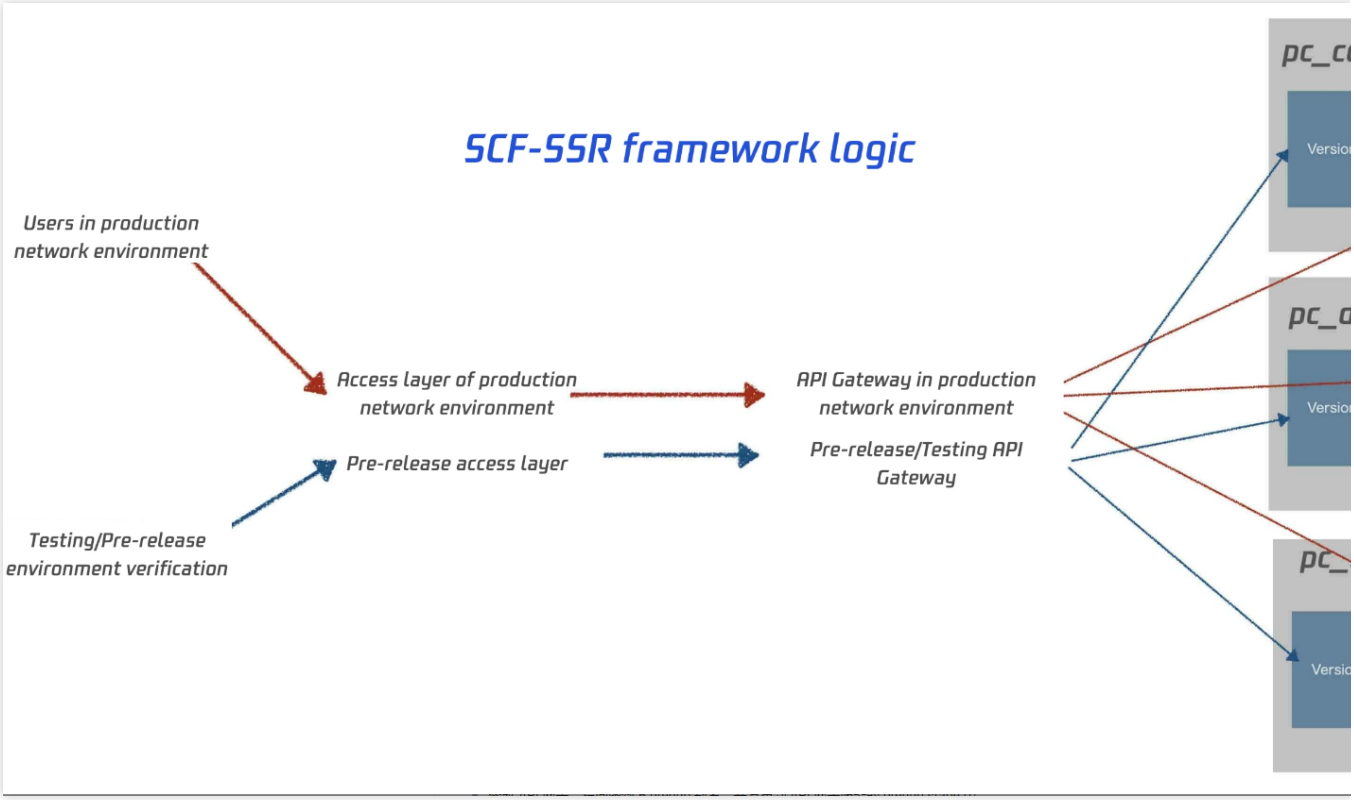
Here, as the current project has implemented automated construction of SCF functions, the corresponding function will be automatically generated with the `.scfssr.json` configuration file, which will have no impact on the current development. Only multiple functions will be generated during the construction, minimizing the application maintenance and development costs.

In addition, based on the SCF function construction process, the code of a single function can be made simpler. Based on the analysis of the dependencies in `package.json`, built-in toolkits can be removed from the function container, and import analysis will be performed on the third-party packages depended on by the function to remove redundancy as shown below:



Problem 2. SCF function release optimization

The logic of SCF-based multi-function SSR scheme is as shown below. When there is a version update, the release process and steps will become relatively complicated.



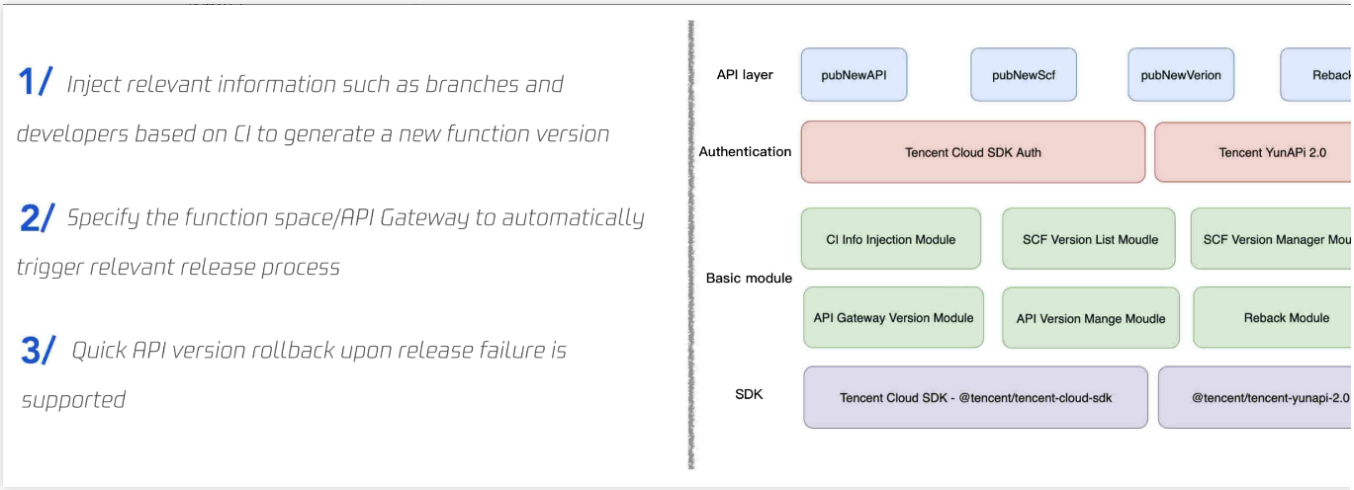
One-Time initialization in configuration

Create the `release` and `prehub` aliases in the function and point them to the `$LATEST` version.
Create service A in API Gateway and configure API Gateway to point it to the `release` alias of function B and publish it into the `release` stage of the API Gateway service.
Modify API Gateway to point it to the `prehub` alias of function B and publish it into the `prehub` stage of the API Gateway service.
Modify API Gateway to point it to the default traffic of function B and publish it into the `dev` stage of the API Gateway service.
At this point, API Gateway configuration is completed, and there will be no need to modify and publish the configuration again in API Gateway.

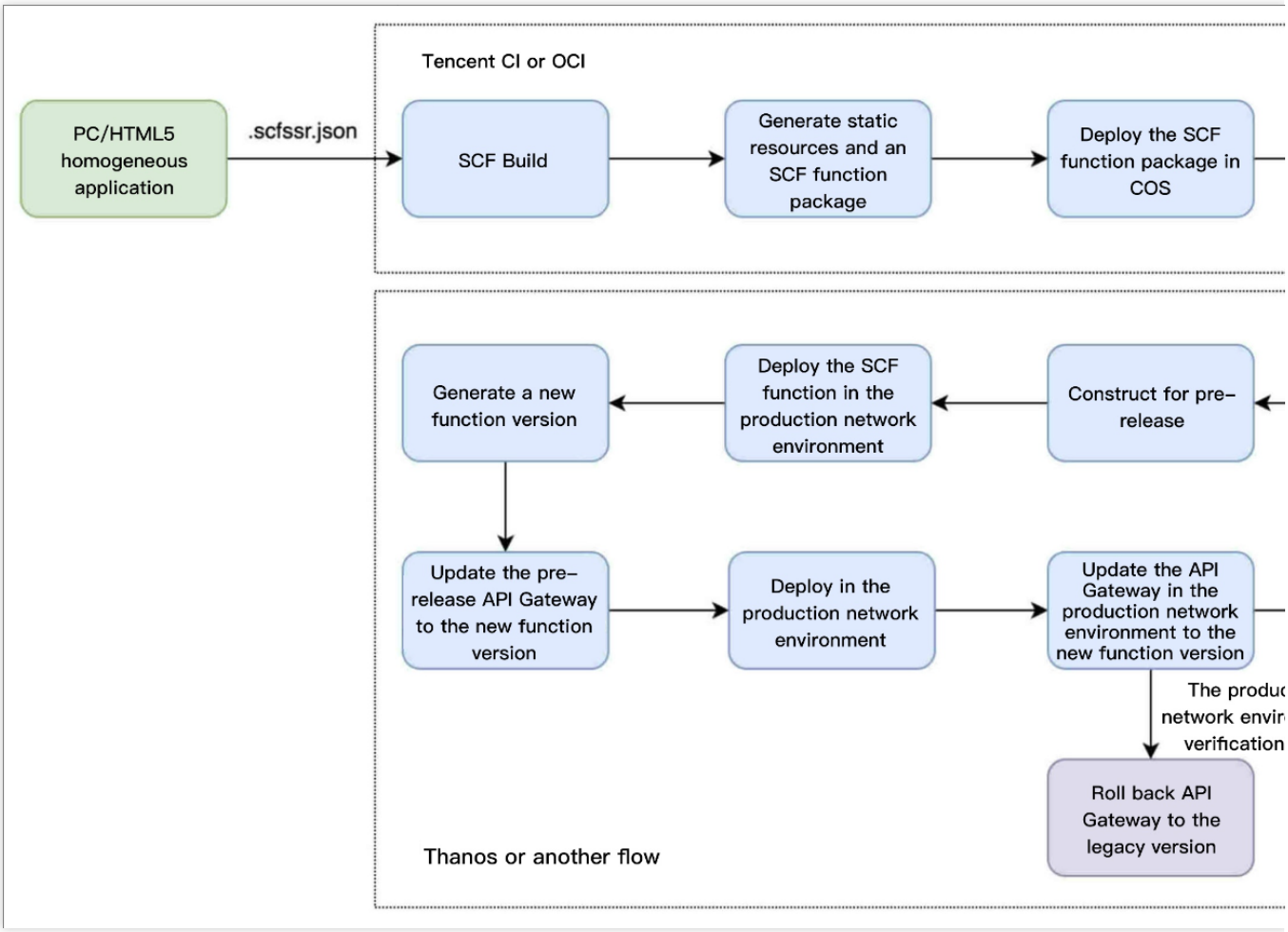
Continuous development, testing, and release

Continuously develop the function and publish the versions 1, 2, 3 at a time.
To develop and test the latest version, configure the `$DEFAULT` alias to point it to the `$LATEST` version. Continuous development and modification can be performed based on this version, and new versions can be published after modification.
The `prehub` alias can be configured to point to version 3 for test and trial in the pre-release environment.
Version 2 has been tried out in the pre-release layer and can be launched. Switch the `release` alias from version 1 to version 2.

View the canary release process through monitoring information and logs and check whether the traffic of versions 2/1 increases/decreases normally, errors of each version during the release, and overall errors.
To optimize the SCF function release process, a quick SCF function release tool developed based on the Node SDK of Tencent Cloud Serverless is provided as shown below:



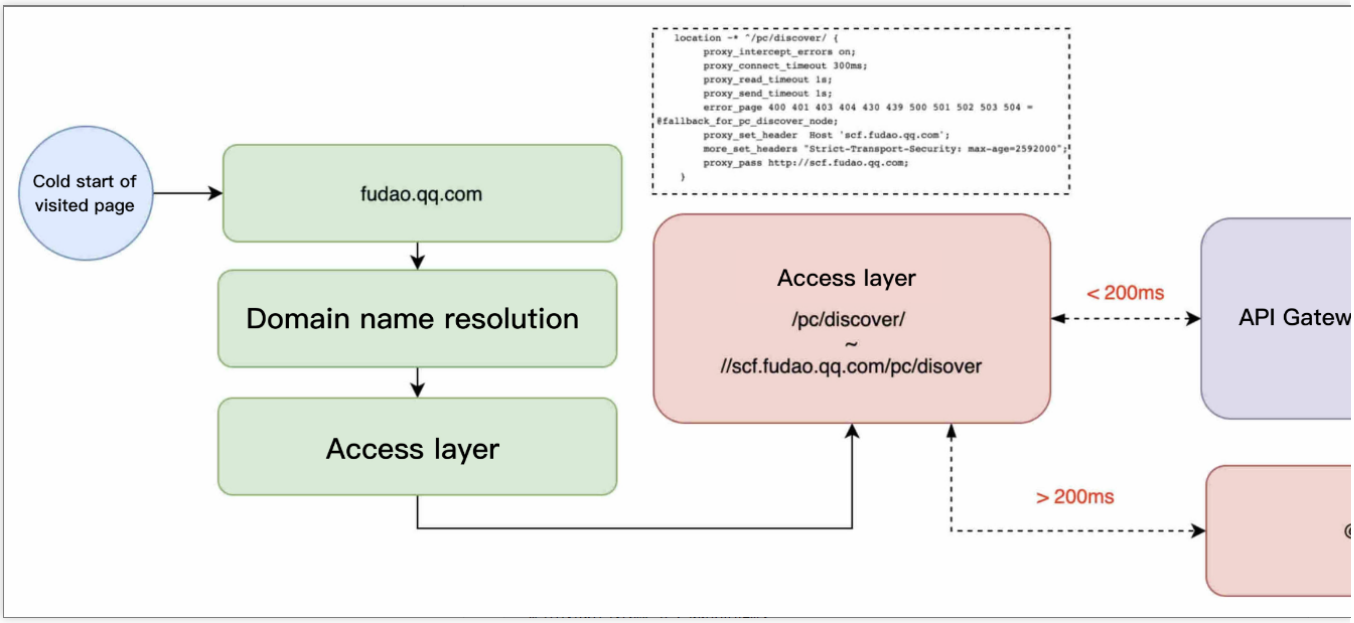
A complete lifecycle of an SCF SSR application is as shown below:



Tencent Cloud Serverless SSR Scheme Strengths and Subsequent Planning

The SCF-based SSR scheme can greatly reduce the service Ops costs. Based on the Tencent Cloud Serverless log system, all individual SSR application requests can have complete linkage on the log platform, which greatly speeds up problem locating and troubleshooting.

The Serverless architecture mode has a long cold start time, and technical improvements have been made in SCF accordingly, such as pre-start container. The actual business can also be optimized; for example, downgrade optimization can be made on the service at the access layer. The optimization is as shown below:



Subsequent optimization can be made from various aspects such as beta test and multi-dimensional downgrade.

Suggestions on Use of SSR Technology

To have a better user experience, we recommend that you perform SSR optimization on core businesses and use Serverless for business deployment and Ops. Serverless reduces your workload of server Ops and scaling, enabling you to improve your team productivity.

After SSR is used, we recommend that you further improve the DevOps process of your business and streamline the entire R&D process, so that development, testing, and deployment can be conducted more efficiently.

We recommend that you use the business access layer for service downgrade so as to improve the SSR application availability.

Online Video Industry
Audio/Video Transcoding Best Practices

Last updated：2024-12-02 16:29:12

Customer Overview

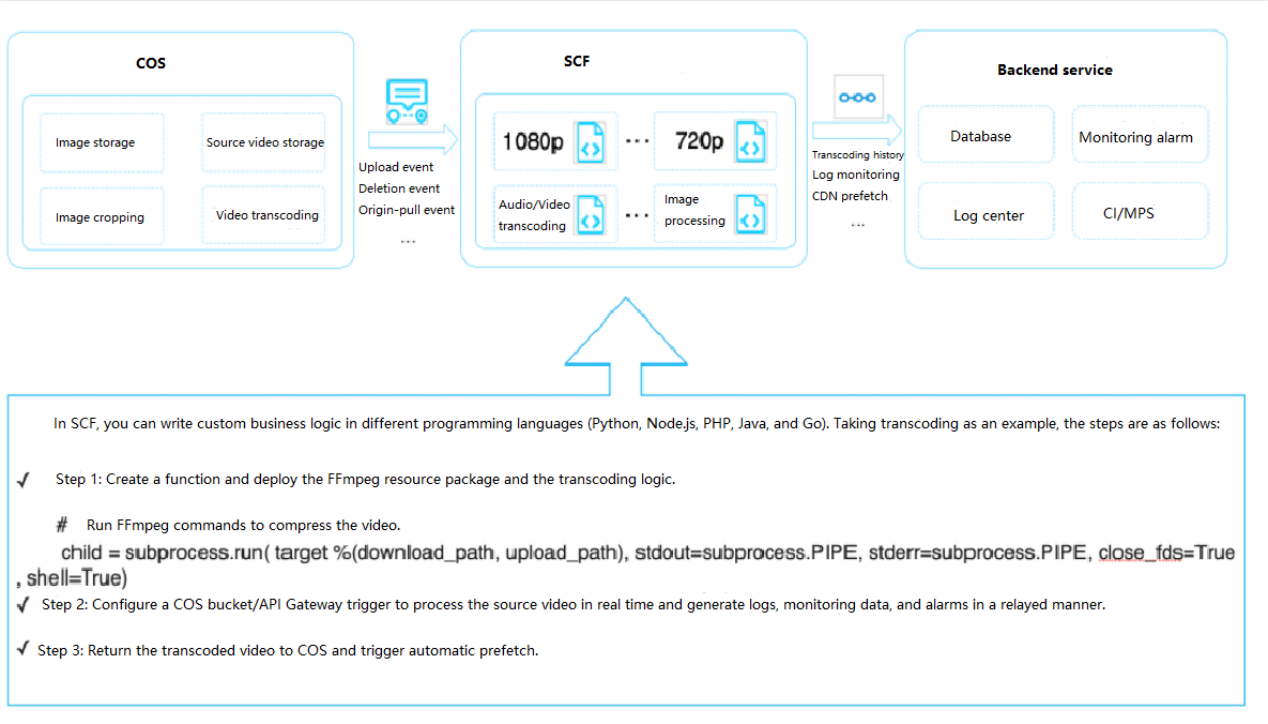
The customer is a comprehensive online education and training group headquartered in Zhongguancun, Beijing, China and listed on the New York Stock Exchange in 2006. Its businesses include foreign language training, K12 education, preschool education, online education, consultancy for studying abroad, and book publishing. It also has several education sub-brands.

Customer Challenges

Every summer, a large number of students study on the customer's platforms. Previously, the storage and transcoding logic for audio/video courseware were implemented in a self-built IDC based on servers and NFS. However, due to the high volume of traffic during the summer, the servers in the IDC might not be able to meet the computing requirements, and it would take very long to purchase new hardware devices for self-built services, so the customer wanted to find a flexible method to support rapid business deployment and complete transcoding efficiently.
In use cases such as video and social networking, users upload high numbers of images and audio/video files frequently, which has high requirements for the real-timeness and concurrency capabilities of the processing system. Traditional container services require the customer to maintain the container clusters on their own and have a poor elastic scalability.

SCF Solution

SCF supports custom transcoding functions to help enterprises quickly build customized task processing capabilities. This makes up for the blind pots of current separate cloud services and conveniently migrates the FFmpeg service from physical servers, cloud servers, or containers to SCF.
SCF, FFmpeg, and COS can be used in combination as shown below to transcode audio/video files:



In terms of technical solution, SCF and COS are used in combination in the cloud to achieve elastic scalability and sustain all local traffic switched to the cloud. In the new business process, a task scheduling module is added to automatically or manually forward the received business traffic to the proprietary cloud-based services. In addition, many high-availability technologies are added to the process, including full-linkage tracking by task `TraceID` and automatic local retry upon cloud computing failure. In the new solution, cloud-based services are easier to develop and maintain and more cost-effective. The pay-as-you-go SCF billing mode also reduces resource costs greatly.

Serverless Application Benefits

Strengths of using SCF to implement the audio/video transcoding service:
SCF provides a standard runtime environment, ensures the high availability and auto scalability of resources, and requires no dedicated maintenance.
SCF is billed by actual usage, eliminating the waste of resources.
Development and debugging in SCF are more efficient, and dependencies are decoupled from the business and can be hot updated separately in real time.
Runtime environments are isolated, so the failure of one single request will not affect the normal execution of other requests.
You need to pay attention to the following to connect your existing business to SCF:
The introduction of SCF needs to be integrated with the existing CI/CD process, so there may be some changes in the development method.

You need to make certain modifications on your existing business code to integrate FFmpeg to the function code and deploy it together with the code file to SCF.

Tencent Online Education

Last updated：2024-12-02 16:29:12

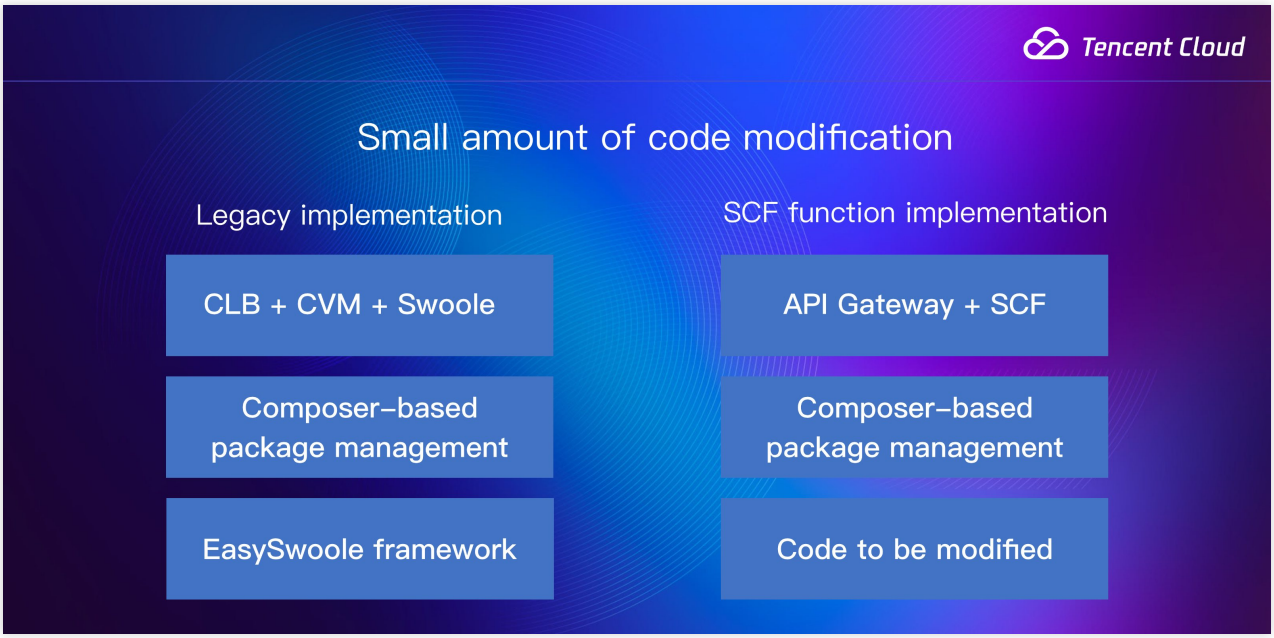
Customer Overview

Social networking is a basic need of game players, which should be implemented in a mature and stable game chat system. This document shows the successful use of Tencent Cloud SCF by RiverGame and describes how to upgrade the chat system in a game with SCF.
RiverGame used its proprietary chat system in its two games: "Top War: Battle Game" and "Farm Town". However, as the number of online players increased, the system stability and costs were facing more challenges, which called for an upgrade in its technology stack so as to reduce the labor and resource costs while ensuring high performance. In this regard, it chose SCF as its solution for the following reasons:
SCF is serverless, which enables the customer to focus on the business logic code while eliminating the need of OPS.
SCF is pay-as-you-go, which avoids resource waste during off-peak hours and reduces costs. It is applicable to small and medium-sized businesses with low complexity such as game chat system API.
Accordingly, it is only needed to focus on how to migrate the existing system seamlessly, i.e., how to use SCF to meet all the specific needs.

Customer Requirements

Requirement 1. Only few code modifications

With Swoole used as the bottom-layer extension, some legacy APIs were deployed in CVM and used CLB to receive external requests, Composer to manage packages at the code layer, and the open-source framework EasySwoole as the HTTP business framework.
With the SCF scheme in place, API Gateway and SCF would be used to provide services at the non-code layer, while Composer would still be used to manage packages at the code layer. The legacy Swoole-based HTTP framework could no longer be used, and the framework was the focus of code modification as shown below:



The following is the modification principle:

1. Determine the logic entry so as to ensure that only one SCF function is used to process all requests.

An entry is actually a route, and a simple route format needs to be defined to get the required information from the function entry code. Then, the information will be transferred to the legacy class for processing, and specific content will be returned. The following is a simple format example of a `url`:

```
https://url/controller/action?query
```

The `controller` and `action` can be obtained simply by parsing the `path` provided by the SCF function. After the fields are determined, the method of the corresponding class will be called to return relevant content. Based on such an entry, the legacy logic processing classes can be called.

2. Process the parent class of the legacy logic processing classes. After the framework is disused, a parent class for basic features, such as getting `querystring` content, parsing `body`, or returning values in a unified format, needs to be implemented.

PHP is used as an example in the following figure, and the principles for different programming languages are similar. The code requires further modification, such as adding database configuration information (which can be passed through environment variables in the SCF function) and async operations for legacy time-consuming tasks.

```
date_default_timezone_set('Asia/Shanghai');
require_once __DIR__ . "/vendor/autoload.php";
function run($event, $context)
{
    $path = $event->path;
    // Parse `path`
    list($controller, $action) = parsePath($path);
```

```
        $controllerClassName = "\\\\App\\\\HttpController\\\\" . ucwords($controller);
        if(!class_exists($controllerClassName)){
            return return404();
        }
        $controllerClass = new $controllerClassName($event, $context);
        // Do not call classes that should not be called externally in the `HttpContro
        if(!controllerClassName instanceof \\App\\BaseController){
            return return404();
        }
        if(!method_exists($controllerClass, $action)){
            return return404();
        }
        try{
            return $controllerClass->$action();
        }catch(Throwable $e){
            return return500();
        }
    }
```

Requirement 2. Quick release

The quick release capability is indispensable. During the migration, various tests were conducted repeatedly. As the local testing feature of SCF did not support PHP during the migration back then, the release process when API Gateway and SCF were used together was as follows:

1. Develop the code
2. Deploy the `$LATEST` version of the function
3. Generate a new version number based on the `$LATEST` version
4. Switch the version in the corresponding path in API Gateway
5. Publish the test version through API Gateway
6. Switch to the version to be used in the production environment through API Gateway

The release process was relatively complicated. When the migration started, API call as described in step 3 was not supported yet; therefore, automated deployment could not be implemented, which is currently supported though. The scheme where API Gateway was used to directly point to the `$LATEST` version of the function and then the function was deployed can still be used, but it is quite simple and only applicable to testing instead of release in the production environment.

We combined the two methods, i.e., using the release process of stable version for stable features and creating an API path pointing to the `$LATEST` version for new features. In this way, new versions could be released at any time without affecting the features in the production environment.

Problem and solution

When API Gateway was published, the resources might exceed the limit, which was found to be caused by the limit on the number of concurrent SCF instances. When a new API version was published, access to a new instance would be

requested while the legacy instance had not been released yet. If the total numbers of new and legacy instances exceeded the limit, it would be needed to apply to Tencent Cloud for quota increase.

Requirement 3. Private network interconnection

The migration was at the system level, but some contents in the legacy system still needed to be used in the future. Therefore, SCF needed to interconnect with the legacy CVM instances over the private network. As SCF supports deployment in the existing private network, it is easy to implement private network interconnection.

Problem and solution

As the API service needed to send requests to the public network, but private network SCF could not directly access the public network during the system migration, NAT Gateway was used to enable SCF to access the public network. For more information, please see Configuring NAT Gateway in VPC.

Note:

When NAT Gateway is used, it is recommended to assign a separate subnet to SCF, as the egress IP will change if NAT Gateway is bound to an existing subnet. If the server IP of the subnet is in some whitelists, it will result in certain impact.

Requirement 4. Log query

Originally, logs were directly stored in the disk and compressed and dumped periodically. After the system was migrated to SCF, the SCF log mechanism needed to be used. SCF can directly deliver logs to CLS, and any output information can be directly delivered as logs. It is required to plan the log contents as needed.

The original information, URL paths, client IPs, parameters after parsing, and business logs of the function entry can be output for quick problem locating and query. In addition, there is no need to output the returned values separately, as SCF can print them automatically.

Note:

After log delivery is enabled, logs can be viewed only after index is enabled. If log content includes index separators, the corresponding keywords need to be deleted from the separators when the index is configured; otherwise, the content will be split.

During the system migration, the SCF log feature had certain shortcomings, such as separate SCF and API Gateway logs. As the original HTTP entry was API Gateway, which might cause some problems hard to be traced. Currently, the `RequestId` fields in SCF and API Gateway have been aligned, which makes it easier to query the logs of the same request through the same ID.

Requirement 5. Time-consuming task processing

The legacy scheme used `task` of Swoole to process time-consuming tasks. As the PHP environment of SCF supports Swoole, the modification scheme in the following figure was used as an initial attempt:

```
$p = "1";
$process = new \\swoole_process(function (\\swoole_process $worker) use ($p){
```

```
    echo $p;
    sleep(3);
    echo $p;
});
$pid = $process->start();
```

However, the process implemented through this scheme could not be completely controllable. As discovered from testing, the printed logs belonged to other requests, and the process time and lifecycle could not be determined. Therefore, a more assured "message queue" scheme was used.

CKafka, Tencent Cloud's proprietary message queue service, was used. After a message body with a general structure is encapsulated and sent to CKafka, it will trigger another SCF function (which runs time-consuming task code). If a general structure is used, the CKafka topic can be ignored. If there is any task requiring async operations, it only needs to be written into the SCF function to be triggered by CKafka, and the name and parameters of the function will be sent to CKafka as shown below:

Problem and solution

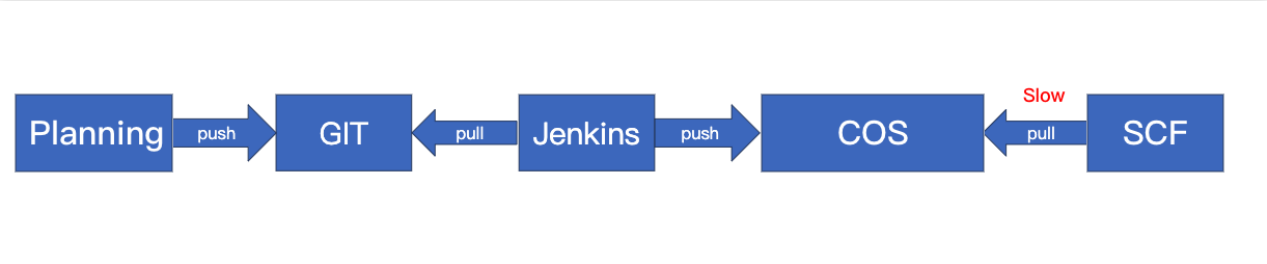
Only one partition is created for one CKafka topic by default. If the consumption speed is not satisfactory, it can be increased by adding new partitions. After a new partition is added, the trigger needs to be deleted and added again in the SCF function.

Requirement 6. Configuration file update

Configuration files in the system were large and required frequent updates, such as the list of banned words used in the chat service.

In the legacy scheme, a configuration file had its own Git library, and Jenkins was started after the planner submitted the file, which then uploaded the file to CVM and reloaded it.

After the system was migrated to SCF, configuration files could not be uploaded separately but could only be placed in the code. The scheme was updated as follows: the planner submits a file to Git, Jenkins gets it from Git and uploads it to COS, and SCF pulls it from COS as shown below:



Problem and solution

There was a performance problem where it took some time for an SCF function to pull data from COS; therefore, files could not be pulled every time when a request was initiated. This meant that the content in each pull needed to be retained in the memory; however, the SCF memory could not be managed uniformly, and real-time change could not be guaranteed.
A compromise scheme is adopted where the pull time of file content retained in the memory is compared with last pull time. If the gap between them exceeds 5 minutes, the content will be pulled again to ensure relative real-timeliness and satisfactory performance and meet the current needs as shown below:

```
private static $fileContent = null;
private static $lastTime = 0;

public static function refresh(){
    self::$fileContent = self::readFromCos("words.txt");
}

public static function getFileContent(){
    if(time() - self::$lastTime > 300 || empty(self::$fileContent)){
        self::refresh();
        self::$lastTime = time();
    }
    return self::$fileContent;
}
```

Benefits to Customer

At this point, all requirements in the system migration had been satisfied, and the migration was carried out smoothly. After migration to SCF, the customer enjoys the following advantages:
The API server does not need to be maintained, and CPU and memory issues can be ignored. Even when the number of requests increases, the customer does not need to consider whether to increase CVM instances.

The monitoring metrics are specific, which can clearly reflect the overall running efficiency through metrics such as access trends and whether there are slow requests or errors.
Decoupling is completely achieved after CKafka is used for message division, which also ensures that no messages will be lost. The method of triggering SCF functions by CKafka is applicable to ever-cumulative slow tasks.
After the version management feature is improved, the versions can be switched at any time, and there is no need to pull code branches for release again.
Tencent Cloud SCF has brought various advantages and benefits to RiverGame, which also plans to use the following features in the future:
Stateless HTTP services, such as customer service message receipt and payment callback APIs.
Async tasks with no returned values needed, such as player ranking reporting in WeChat Mini Game.
Scheduled tasks, such as periodically pushing relevant event information to players.

Best Practice of Tencent IEG Going Global

Last updated：2024-12-02 16:29:12

Customer Overview

Tencent Games under Interactive Entertainment Group (IEG) focuses on game development and operations and maintains major online game communities. On its journey to cloud gaming, IEG has been constantly driving and making breakthroughs. In March 2021, IEG launched an online game development platform called "Proxima Game Online Service" (PGOS) as part of its global business, which provides:

Backend service platform: PGOS is an online game solution designed to facilitate game backend development and maintenance and reduce costs, so that developers can focus on the development of gameplay and core logic.

Fully managed service: its complete backend solution eliminates the challenges in setting up, managing, and running servers at scale. Its automatically scalable dedicated servers provide a low latency and high reliability for real-time gaming.

One-Stop control panel: developers and OPS personnel can retrieve player information, view logs, monitor real-time data, and edit service configurations on the its web portal.

Cross-platform SDK: it provides out-of-the-box SDKs for C++ and UE4, making it easy for developers to use the PGOS service on game clients and internal servers.

Flexible matching rules: it enables players to accurately and quickly match up with other players in real-time battles.

Scalability and flexibility: its entire system uses a microservice architecture rather than integrated hierarchical architecture, allowing developers to add and modify interactions between corresponding services.

Serverless Solution

Tencent Cloud Serverless naturally supports the above features provided by PGOS and is used by it to provide underlying computing support, better helping teams migrate to the cloud quickly.

Out-of-the-Box serivce: Tencent Cloud Serverless enables you to focus entirely on business code with no need to purchase, set up, and configure servers. Its architecture not only helps accelerate game publishing and iteration but also significantly reduces the OPS costs. In addition, it guarantees the stability and security of your business and the availability of the resources, eliminating your concerns over underlying resources.

Dynamic scaling: another benefit of serverless is auto scaling, which makes it easier for your business to withstand traffic surges. Auto scaling guarantees normal business operations when access traffic surges and reduces costs during off-peak hours.

Real-Time monitoring: Tencent Cloud Serverless offers real-time logs and a monitoring dashboard, where R&D and management personnel can monitor business operations in real time. In addition, it is connected to Cloud Monitor to

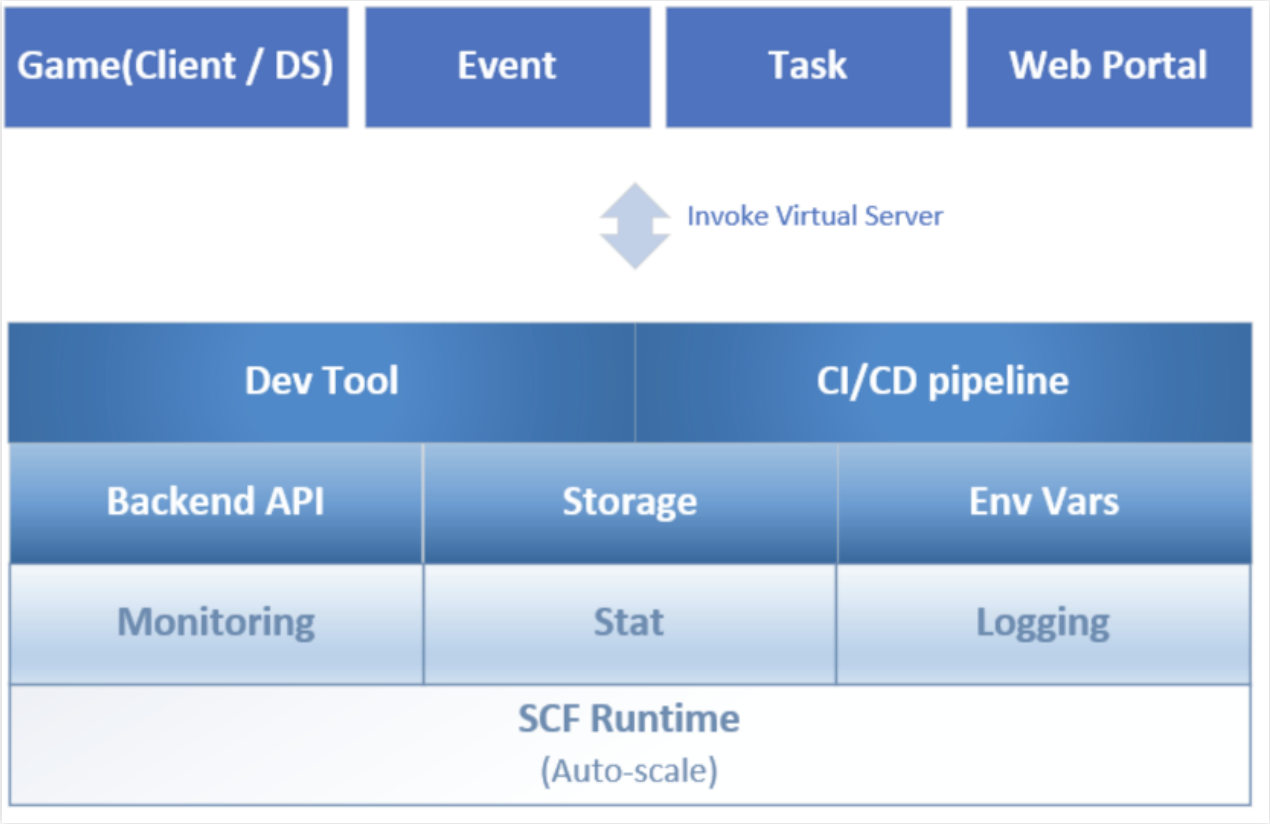
provide alarming capabilities in multiple dimensions such as execution duration and status exception. In this way, you can discover problems and get notified as soon as they occur.

Scalability and flexiblity: the atomic feature of FaaS naturally supports flexible business expansion. Different SCF functions can support independent features, such as mutual function invocation, separate update and deployment, and online function code editing. Tencent Cloud Serverless offers a one-stop solution from business development, deployment, to monitoring.

Multiple event triggering methods: Tencent Cloud Serverless supports around 10 event triggering methods, including timer trigger, API Gateway trigger, and COS trigger, meeting your diverse needs in different scenarios.

How serverless supports cloud gaming with computing power

Tencent Cloud Serverless can provide underlying computing support for the global business of PGOS. You can create one virtual server (corresponding to one or more SCF functions) and write relevant business logic. PGOS relies on it to offer complete monitoring and logging capabilities and connect to backend services. It further encapsulates DevOps tools and furnish fully managed and automatically built and deployed features.



PGOS provides multiple drive methods, and the underlying layer triggers business operations on the virtual server with different function triggers.

Timer drive: a scheduled task can be configured for a game on the web portal to trigger a specific API of the virtual server.

Event drive: the virtual server can listen on specific events and will be automatically triggered when events occurs.

*_Game drive_: the game client or DS can actively invoke extension interfaces and trigger specific APIs of the virtual server through the gateway.

Manual drive: you can manually run/trigger the specific APIs of the virtual server in the web portal.