ZUMOBI

Zumobi Brand Integration(Zbi) Platform Architecture Whitepaper

Table of Contents

Introduction 2
High-Level Platform Architecture Diagram 3
Zbi Production Environment 4
Zbi Publishing Engine 5
High-Level Feed Processor Architecture 5
Crawling Service Architecture 6
Parsing Service Architecture 6
Scheduler Architecture 7
Image Service Architecture 8
Video Service Architecture 8
Content Pipeline Monitoring Architecture 10
Zbi Platform Security Overview 12
Client 12
Server 13

1525 4th Avenue Suite 800, Seattle, WA 98101
Zumobi.com

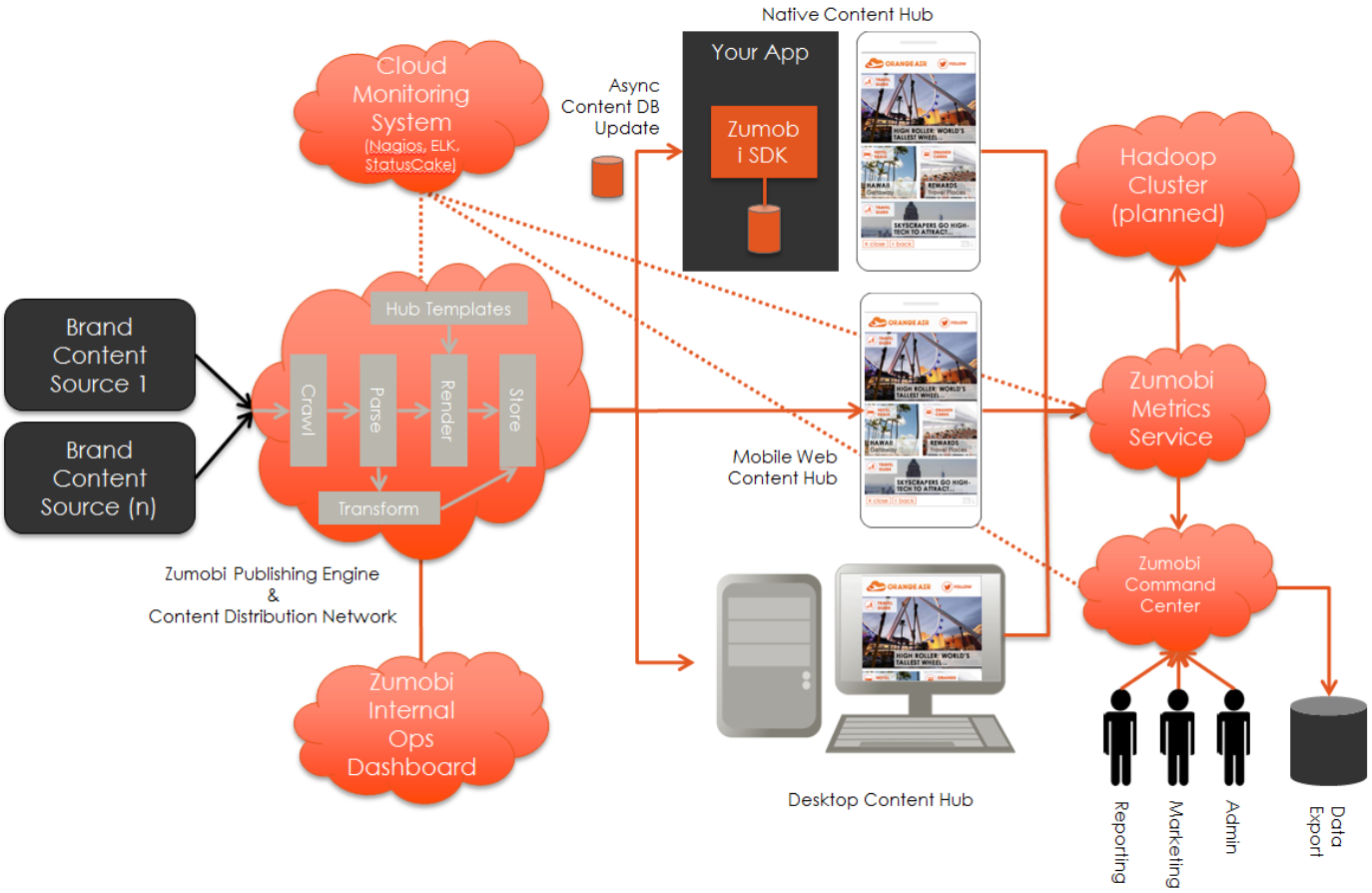
Introduction

The Zumobi Brand Integration (Zbi) Platform represents the next step in the evolution of the Zumobi Mobile Content Marketing strategy by enabling the marketing teams of major mobile applications (banking, automotive, etc) to communicate directly with their end users using proven Zumobi content distribution concepts in a seamless and secure manner. Essentially Zbi is a platform that allows the owners of widely installed mobile applications to create a private "Content Hub" that the in-house marketing teams can utilize to communicate specific content, promotions, etc directly to their app user base in a trusted fashion.

The Zbi platform is comprised of 3 main subsystems:

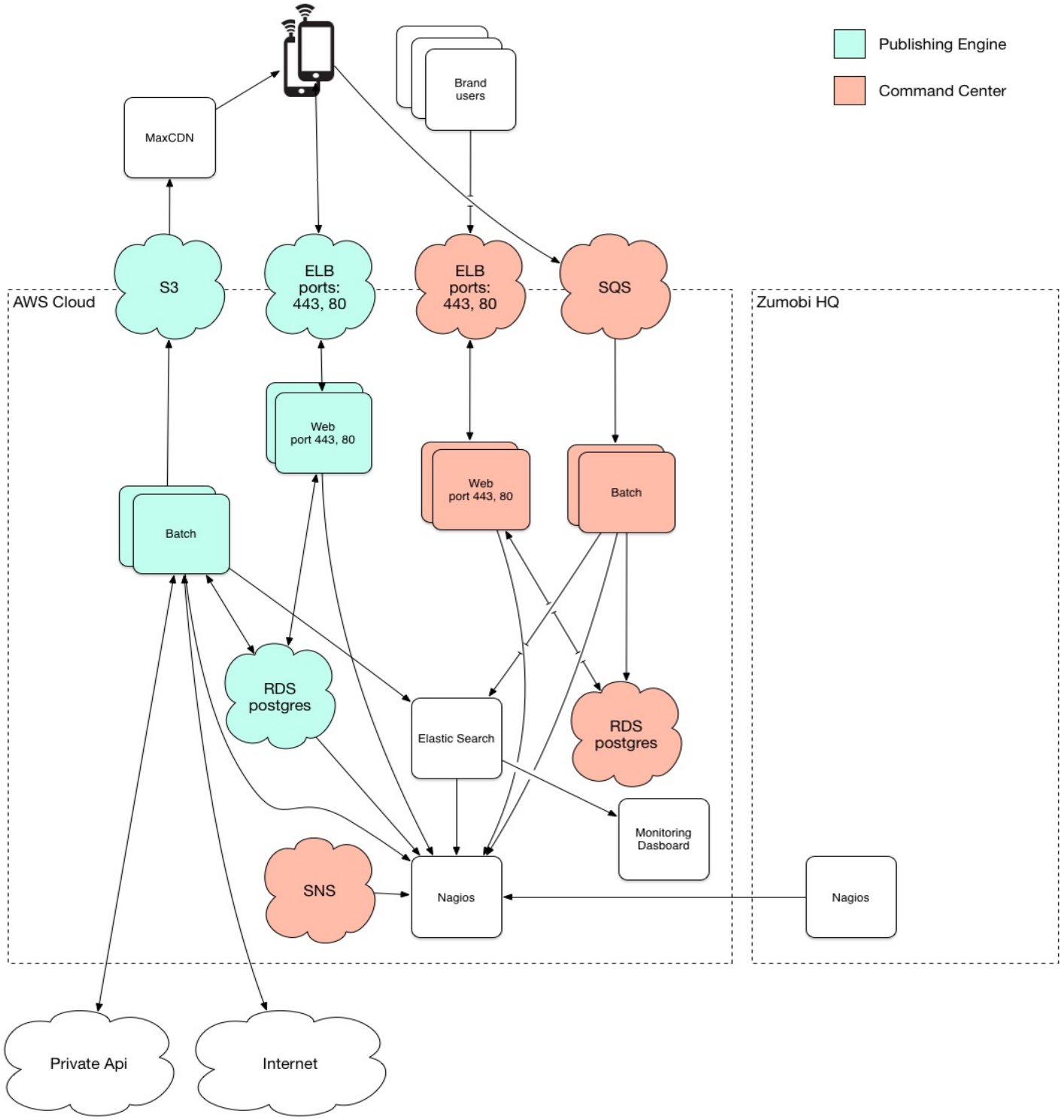
1. The cloud-based Zbi Publishing Engine which is responsible for automated processing, tagging, rendering, packaging, encryption, distribution, and monitoring of the Content Hub(s) at scale.
2. The Zbi Command Center which provides marketers the ability to generate relevant Content Hub reports on various performance dimensions captured by the SDK in aggregate using a self-serve interface.
3. The native client SDK that is designed to be integrated into the marketer's native application which enables the application to securely request, download, verify, cache, and display the Content Hub within the application. The SDK also allows the application user to interact with the Content Hub within the native application while offline with full metrics capture capabilities.

High-Level Platform Architecture Diagram



ZUMOBI

Zbi Production Environment



1525 4th Avenue Suite 800, Seattle, WA 98101
Zumobi.com

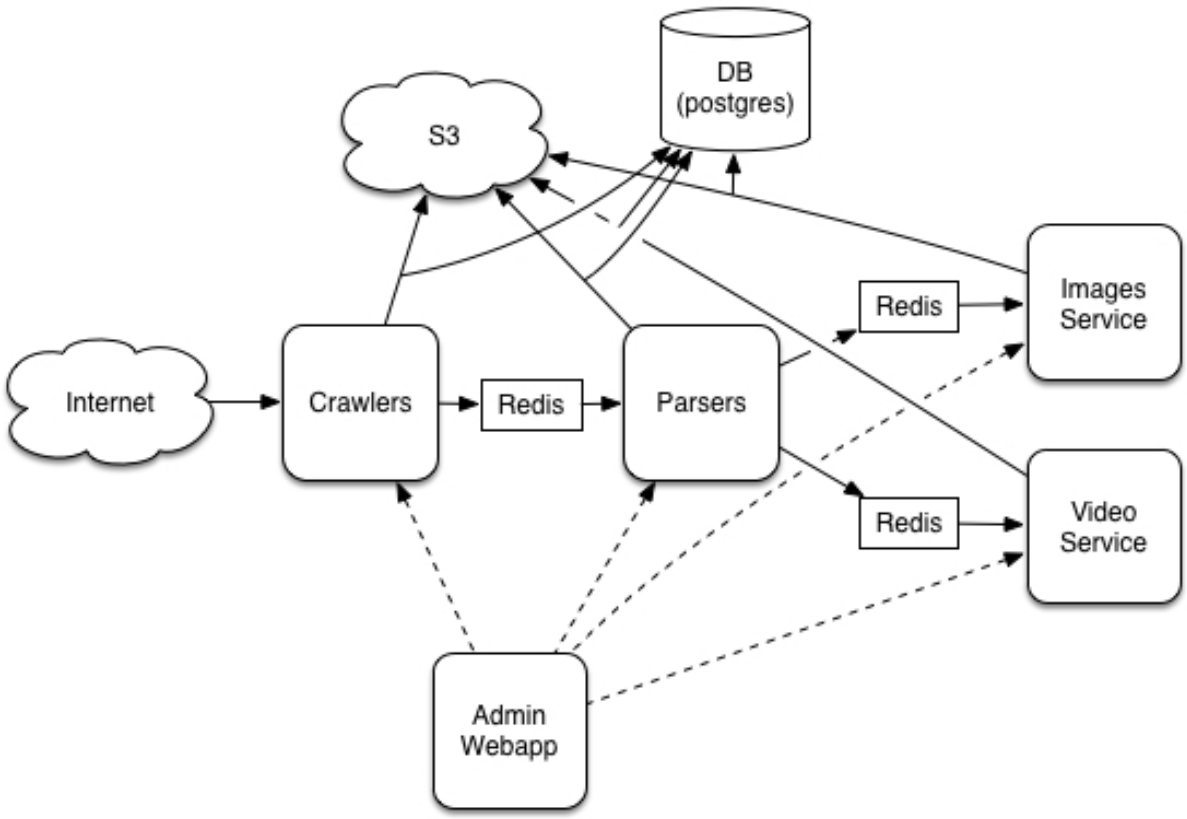
Zbi Publishing Engine

The Zbi Publishing Engine is a cloud-based system designed for fault-tolerant content feed processing and aggregation of metadata. The core of its functionality is to fetch content from a variety of structured content sources (RSS. Atom, JSON, etc), parse it according to business rules enacted through code or through configuration, transform/optimize the content for mobile displays and distribution, and render the final result using a set of content templates designed specifically for the brand's experience into a micro content DB.

The Zbi Publishing Engine is comprised of 6 primary software components:

1. Crawler Services
2. Parser Services
3. Scheduler
4. Image Service
5. Video and Transcoding Service
6. Render Service

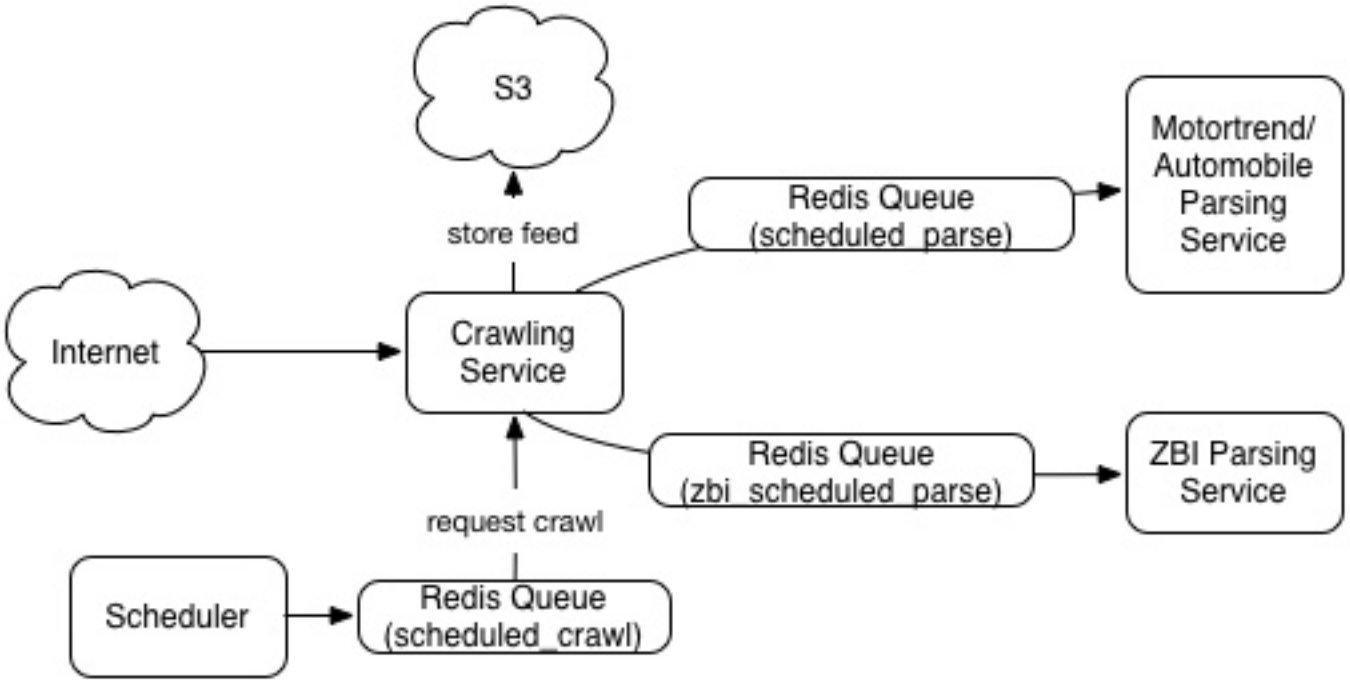
High-Level Feed Processor Architecture



- Crawlers: Fetch content feeds from brand API's

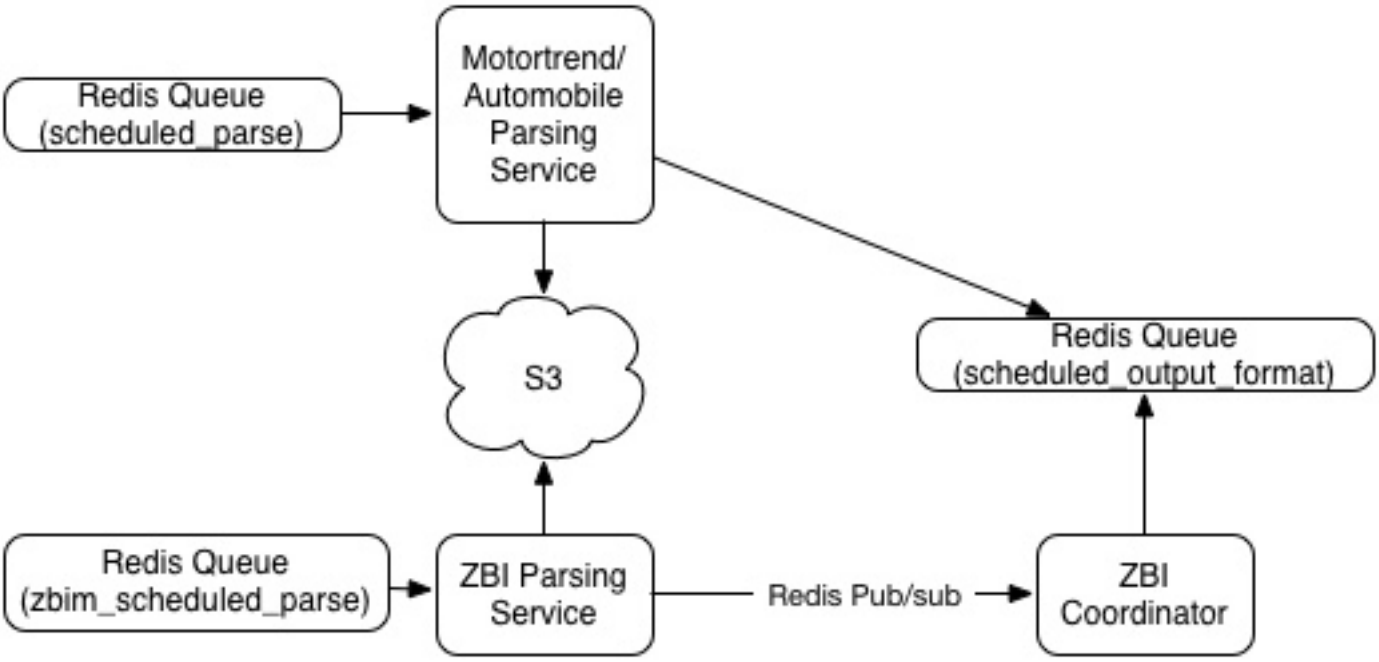
- Parsers: Parse the feed and extract the content as well as images/videos links for later processing
- Image Service: Download the linked images to S3 and also serves as a processing service (resize)
- Video Service: Download Videos to S3 and perform set of transcoding for device optimized videos
- WebAdmin app: Monitor the pipelines. Manage the feed creation
- Redis: Queueing system to communicate between services
- DB: Store the relationship and details about feed, links, images, tags, etc.

Crawling Service Architecture



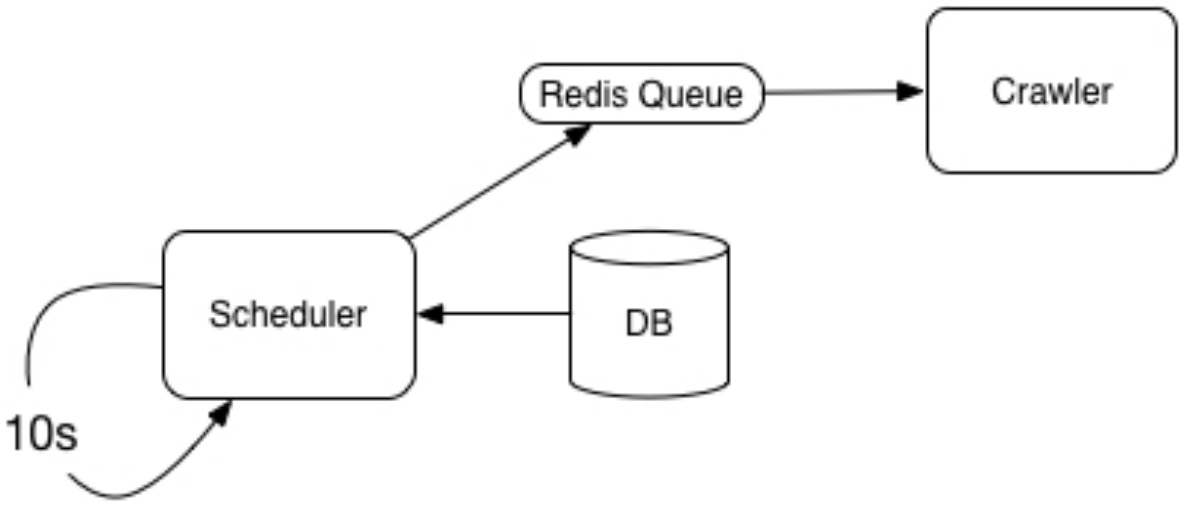
- **S3**: Storage of the downloaded crawl. See next section for details
- **Scheduler**: Schedules Feed for download based on a configuration and submit a request for crawl to the crawling service via redis queue
- **Zbi Parsing**: Parses the feeds

Parsing Service Architecture



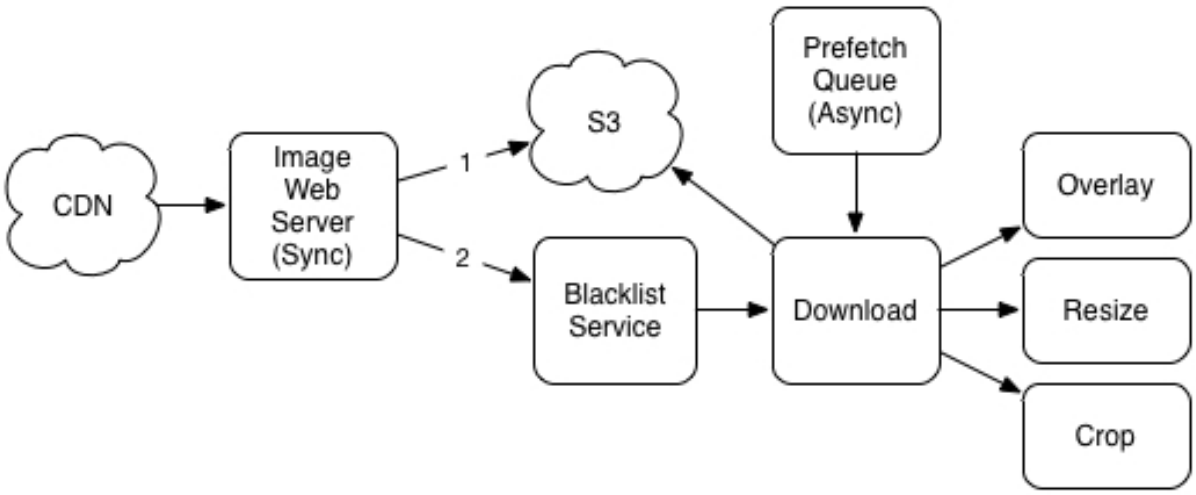
- **S3**: Storage of the parsed images for Zbi parsing crawl. See next section for details
- **Zbi Parsing**: Parses the feeds but requires downloaded images
- **Zbi Coordinator**: Coordinate the feeds generation based on their parsing status submitted via a publish/subscribe mechanism using Redis pub sub

Scheduler Architecture



The scheduler allows for scheduling certain type of jobs such as crawling jobs for the feeds. This is really just a simple piece that create a set of tickets for crawling (FeedCrawlingRun) and queues them for processing in a Redis queue . This enables an automated recurring scheduler or manual scheduling of a crawling. (Just inject the ID into the queue for processing.)

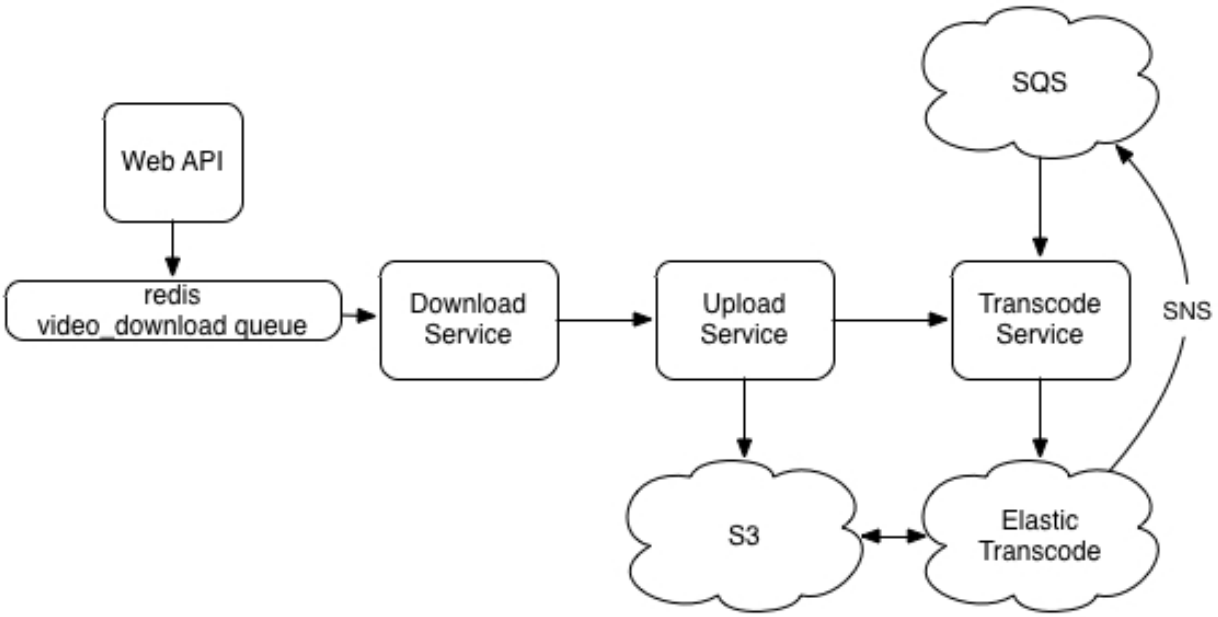
Image Service Architecture



Process for Image Request: An Image request is sent to our image web service (either by the CDN or directly by a user.)

1. It will check our cache if the image exists for that specific size. If not, the full service is ran
2. It will try first to download the most recent version of that image from our partner, resize it and then cache it to S3
3. In the event the partner site is down or the image is removed, we will look up our S3 cache of the last Original we ever downloaded and resize using that image
4. If the image can never be downloaded and found in our cache either, we will return a 404

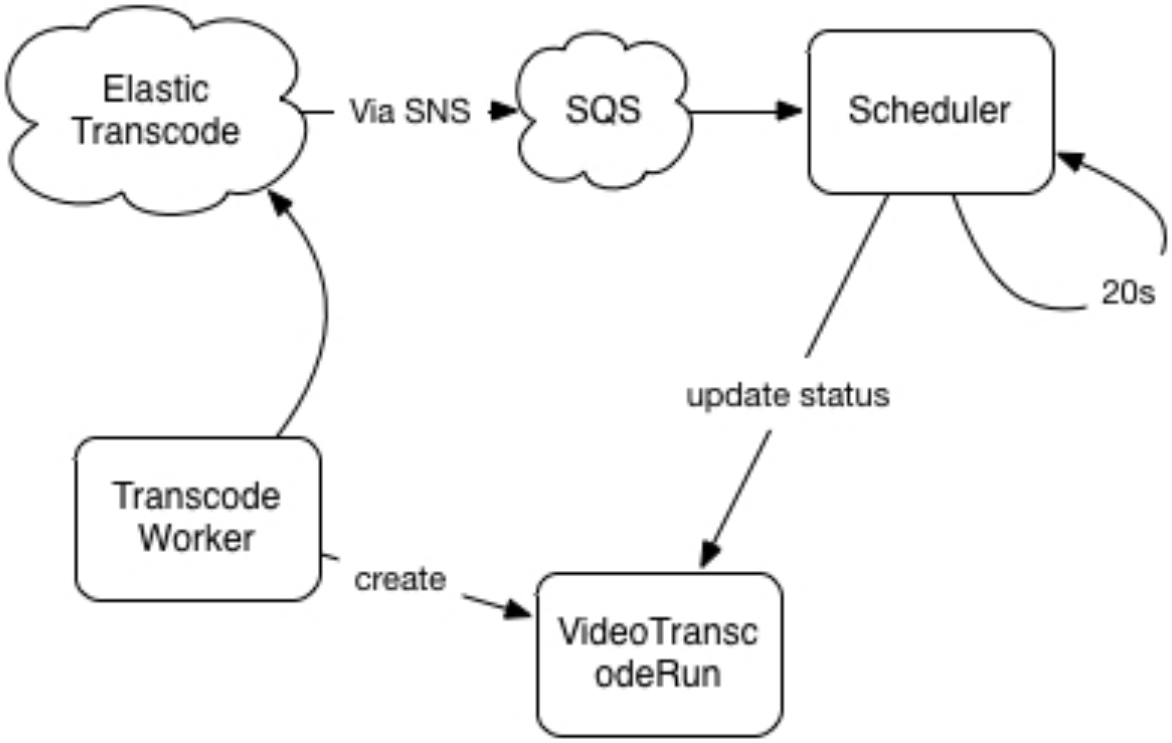
Video Service Architecture



- S3: Storage of the video during the upload phase. Also storage for the ticket specific needs such as backtrace or any other files needed to be saved
- SQS/SNS: Status result of the transcoding job is submitted asynchronously via SNS to a SQS queue
- Redis: Use as queuing mechanisms to send message asynchronously from one part of the pipeline to the next
- Download Service: Perform the download of the files from internet to the local disk
- Upload Service: Perform the upload of a local video to S3 video in bucket
- Transcoder: The Elastic Trancoder from AWS. Currently used via a pipeline where you submit a job for a specific key in the video in bucket(configurable). Output will be going into a video out bucket (configurable)
- Web API: A simple API exposing the video service for easy integration with other web apps. You can submit a url and perform the status check on that request all the way till it is done or errored

Video Transcoding Listener

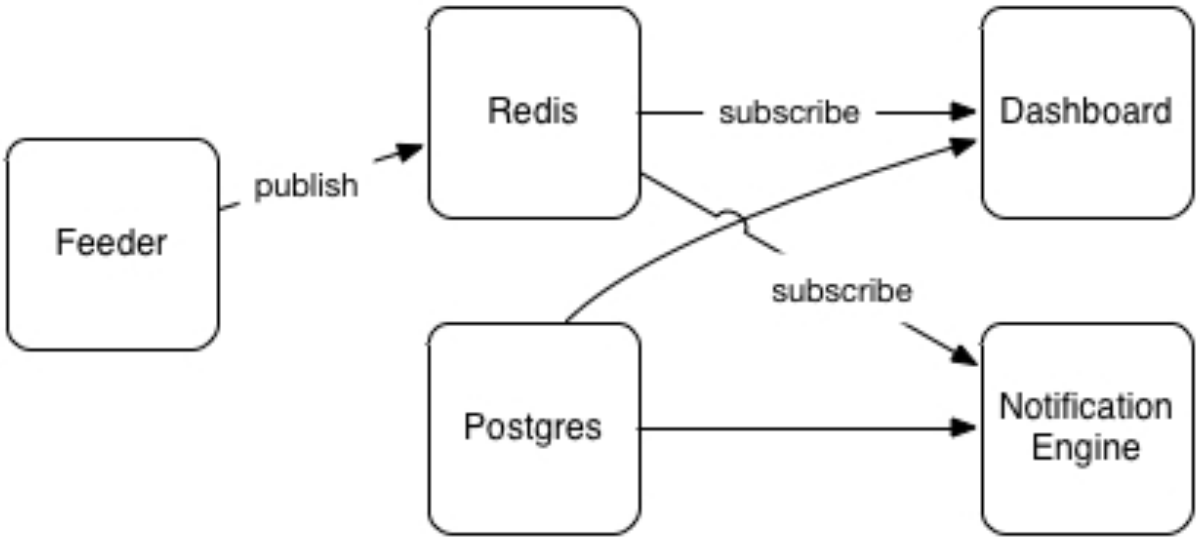
Video transcoding can take many hours. To prevent blocking and waiting, we instead have a scheduled job that runs (20s by default) to check a SQS queue, where SNS message are being dropped on certain events. When the video for example is done transcoded, a SNS message is sent to the queue with the actual status of the job. So we have a job that is scheduled to check on that queue, dequeue the message and figure out how to update the VideoTranscodeRun status based on that message.



Content Pipeline Monitoring Architecture

The Feed Publishing Engine (Feeder) publishes events to the Redis PubSub on specific topics (Crawl, parse, output generation, etc..) Then the set of subscriber can listen for events. The notification engine allows to set rules such as repeated errors (aggregate similar errors) over a period of time and perform notifications (SMS, email, etc..) as needed. It also uses postgres to reconcile specific errors details and potentially submit the details in the notifications (like backtrace, etc).

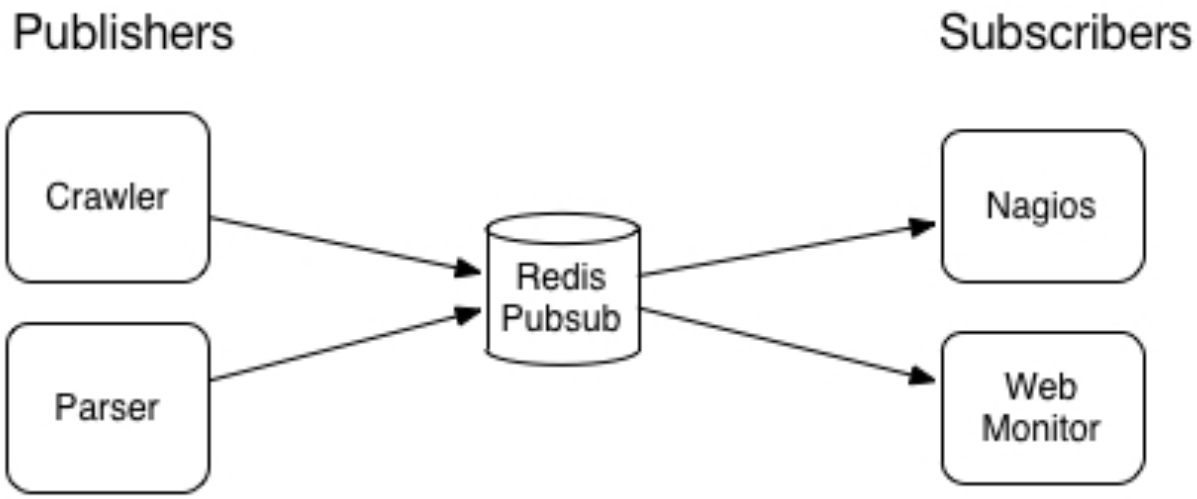
The Zbi Command Center dashboard provides an easy interface to see live events as well as trends over a period of time to analyze or perform historical analysis on events. It can also make use of the postgres database to get specific details of stored tickets (FeedCrawlingRun, FeedParsingRun, etc..)



Content Processing Statistics

In order to detect abnormal service performance, our platform collects various service statistics such as crawling speed, parsing speed, etc.. These statistics are used for building real-time reports and for troubleshooting service issues. Statistics will be stored as part of the ticket as a JSON format. Each of the Runnable model has its own table in the DB (these tables have “Runs” in their name) will have a hstore field to collect the statistic within the PostgreSQL database

However, if the database goes down, we would not be able to know the current status of a crawler or parser. In order solve for this potential issue (and to keep the design simple), a real-time monitoring subsystem is needed to complement the database so we can really see what is happening live on our systems. In order to satisfy the both the real-time monitoring requirements and the ability to see the state of each crawler and parser, we leveraged the pub-sub mechanism supported by Redis. This mechanism is very fast and makes it easy to plug many subscribers that want to listen for events. We could even push such events to a data warehouse for archival purposes without impacting the overall performance of the system.



Platform Security Overview

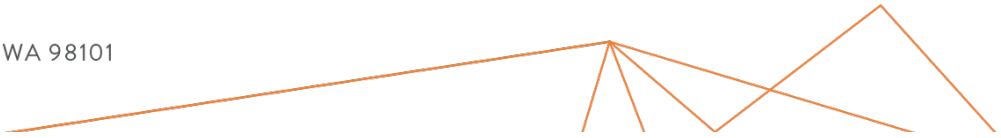
Client

- Trusted Transport
    - The content bundle that is produced by the Zbi Publishing Engine is delivered to the mobile application over https with TLS certificate validation enabled to ensure that the content bundle is downloaded from a trusted source
    - Once the bundle has been securely downloaded, the Zbi SDK will verify the bundle's integrity using a secured checksum generated by the Zbi Publishing Engine to ensure that the contents of the bundle were not tampered with during transport
    - If the content bundle fails the checksum test, the Zbi SDK will automatically discard the results and fetch the content bundle over the secured transport again
    - If the Zbi SDK fails to receive a valid content bundle after three tries, the Zbi SDK will enter into a safe mode to prevent the use of content that is not secure.
- Secured web view
    - All non-video assets for rendering the content within the Zbi web view are included in the content bundle. All links contained in the content point to pages/assets are contained within the content bundle. No external links are allowed
    - Video content can be streamed into the application using Zumobi's secured video transcoding and hosting infrastructure or directly from well-known sources like YouTube
- Remote kill switch
    - If a critical vulnerability is ever discovered in the Zbi SDK, the SDK has a built in mechanism that can be triggered remotely using the Zumobi secured infrastructure to disable the app's usage of the SDK and the content bundle. In addition, the SDK will purge any existing content bundles to guarantee a clean start scenario once the SDK functionality is re-enabled through an

app update
- Once disabled, the app will no longer be able to use the functionality provided by the Zbi SDK until the app is updated. As part of the app update process, the flag that controls whether or not the SDK is disabled would be reset to enable the use of the SDK with the app
- The command issued to the app to disable the functionality of the SDK is only allowed to be transmitted over the same trusted transport as the content bundle

Server

- Intrusion detection system
    - Zbi server infrastructure components, such as the Zbi Publishing Engine, are automatically monitored for any suspicious activity that would alert key personnel if critical systems or files were compromised
- Authentication and authorization
    - All access to Zumobi server systems are only allowed through named user accounts centrally managed through a secured key management system based on unique private/public key pairs created explicitly for each user account
    - Password authentication is explicitly disabled to prevent brute-force/dictionary attacks
    - Remote administrator account access is explicitly disabled and such privilege escalation is only granted on an as-needed basis using explicitly named accounts managed through a centralized secure configuration system. Activities performed by a user granted such rights are logged for future reference.
    - Each Zbi server application uses its own assigned application account, which limits the application to only the resources that it has been allowed to access
    - Credentials for server resources that are not able to be secured using a key-based approach are protected using a secured credential management system that is only accessible to administrators
- Network access
    - The Zbi production environment only allows public access to its protected resources over port 443. Our infrastructure is also capable of filtering access by source IP or a specific user agent.
    - Shell access to the Zbi production environment is only allowed over the Zumobi virtual private network that is safeguarded by a Radius authentication system
    - All non-essential network resources (RPC interfaces, etc) are explicitly disabled
- Centralized, secure logging infrastructure
    - All Zbi server applications write logs to restricted directories that are only accessible to the application and administrators
    - Logs are shipped off the server to a central, secure log collector that provides controlled access to the application and systems logs
    - One year of application and system logs are kept on the collector for quick access. Older logs are archived for long term storage
- Change management
    - All Zbi servers and applications are managed through a secure central configuration



management and deployment system that track all changes made to the production environment and controls access to sensitive production configuration information
    - Code destined for release into the production environment is hosted in a secure package/artifact store that is only accessible to administrators and fetched by the automated deployment system when needed
    - Code developed for use in the production environment is peer-reviewed and tested using Zumobi's automated testing framework to ensure a high level of quality
    - The source code repository for the Zbi Platform can only be accessed using key pairs uniquely created and assigned to each developer
- Monitoring
    - The entire Zbi production infrastructure is monitored automatically through a centralized internal system that is only accessible to secured administrators
    - Server components instrumented for monitoring purposes only expose those interfaces to the internal monitoring system. Users or applications external to the production network (eg: via the internet) are unable to access these interfaces
    - Monitoring data is collected by the central monitoring system and kept for up to one year