# ZenScript

Zendesk Inc.

## What Is Zendesk?

Zendesk delivers the leading cloud-based customer service software. Loved by customers for its simplicity and elegance, Zendesk is the easiest and fastest way to provide great customer service. Our solution is easy to try, buy, implement, and use.

We are a midsized company located in the heart of San Francisco. The company was founded by 3 friends in Denmark, who still serve as CEO, Chief Technology Officer and Chief Product Officer of the company. After a successful IPO in May 2014, we now have over 50,000 paying customers and we just hired our 1000th employee.

Zendesk is a "B2B" (Business-to-Business) company: our customers are themselves businesses, who need a customer support solution that enables them to efficiently provide support to their own customers.

We offer our product on a Software as a Service (SaaS) basis. Customers do not need to install anything on-premise, and need nothing more than a Web browser to access Zendesk. Zendesk is also a multi-tenant architecture: All customers are served by the same (large) number of servers. Customers don't get their own "instance" of Zendesk running on dedicated hardware, unlike the way much traditional enterprise software is delivered.

## What Does Zendesk Do?

Consider a hypothetical company, CoolCo, that uses Zendesk. CoolCo's own customers are referred to as "end users" in Zendesk terminology. CoolCo has a Support team staffed by support agents who use Zendesk's agent user interface.

End users request support from CoolCo in a variety of ways:

- By emailing `support@coolco.com`,
- By clicking "Live Chat" on CoolCo's website,
- By filling out a support form on CoolCo's website
- By tweeting to `@coolco` or posting on its Facebook page,
- Or by calling CoolCo's phone number.

All of these avenues to support are referred to as channels, and all of them are funneled into Zendesk to create tickets. A ticket in Zendesk is essentially a conversation between a customer and a support agent. All of the above channels create tickets in Zendesk. Tickets are requested by customers or on their behalf, and then solved by support agents using Zendesk's agent user interface.

Zendesk is, at its heart, a ticketing system. "Ticket" is the core data model object in Zendesk. Tickets are trackable items with persistent identifiers that can be assigned to different people and otherwise moved through some type of workflow. Zendesk is a ticketing system focused on customer support. Other examples of ticketing systems are Atlassian JIRA, which is focused on project management, and Bugzilla, a bug-tracking system.

## Business Rules

Zendesk features business rules that enable behavior to be associated with changes to tickets or when certain events occur over time. These rules offer a sophisticated set of conditions for when they execute, and actions that they can take.

Large, sophisticated customers of Zendesk often have hundreds or even thousands of business rules in their Zendesk instance. Many of these rules often turn out to be quite similar. Because Zendesk business rules are not a full-fledged programming language, logic that could be expressed concisely in a programming language needs to be "unrolled" repetitively into many copies of the same thing with slight variations.

In addition, Zendesk has a feature called Targets, which enables a customer-designated Web service to be invoked when a business rule is triggered. The customer Web service might then use the Zendesk API to make further changes in Zendesk. Often, it might be more convenient and efficient to have such code running directly inside of Zendesk.

## Introducing ZenScript

The idea of ZenScript is to enhance Zendesk's business rules system and Targets system, and perhaps other areas of the product as well, with a lightweight scripting language that can access Zendesk's core functionality and model data in a safe, secure, and efficient way that is developer-friendly.

This is a unique opportunity, in the sense that ZenScript, if it came to fruition, could become an essential foundational technology of Zendesk's platform. Zendesk has 50,000 customers, and many of the most important and sophisticated ones would surely make use of this technology. Zendesk would also make heavy use of it internally... it could become a fundamental way that Zendesk is extended to do new things.

## Zendesk Team

Gary Grossman (`ggrossman@zendesk.com`)
Director of Engineering

Gary will be the key point of contact at Zendesk and lead technical adviser for the project. Prior to Zendesk, Gary spent many years working at Macromedia, later acquired by Adobe Systems, and was the original developer of the ActionScript programming language used within Adobe Flash. Gary has done a lot of the above in a different context: developing code editors, debuggers, compilers, virtual machines, and API layers. Gary's past experience with ActionScript can hopefully be leveraged to help the Harvey Mudd team drive this project forward.

Steven Yan (`steven@zendesk.com`)
Director of Product Management

Steven is a Harvey Mudd alumnus and has been at Zendesk since 2009. He leads Product Management in many areas of Zendesk, and in particular leads the Zendesk Platform agenda, specifically the Zendesk API, App Framework, and App Marketplace, all of which are highly relevant to the ZenScript project. Steven's long experience with Zendesk gives him an unparalleled depth of knowledge in the product and business, and he may be consulted to give insight into these aspects of the project.

# Milestones

This project is a complex undertaking, so it is helpful to break it down into a road map of smaller milestones:

- Product Training
- Scope and Use Cases
- Technology Selection
- Runtime Proof of Concept
- API Layer
- Security Sandbox
- Container Deployment
- Code Editing and Debugging Experience
- Documentation, Tutorial, and Examples

## Product Training

Zendesk sounds simple enough: a ticketing system for providing customer support. However, it's a product of surprising depth, and a thorough understanding of it will be required in order to execute on this project. Zendesk has excellent product training resources, and we should start this project by giving the Harvey Mudd team a crash course in Zendesk and get them using the product so that they can clearly understand what they are trying to build.

## Scope and Use Cases

Before we launch into writing code, the Zendesk and Harvey Mudd Clinic combined team should sit down and brainstorm about the use cases for this new subsystem of Zendesk. What customers will use this functionality, and to what ends? Further, given that the project will go on for many months but many months is not long in software development terms, it will be critical to decide on a scope that is doable in the Harvey Mudd's team timeframe, and be vigilant about what features are must-haves vs. nice-to-haves. Changes and course corrections will undoubtedly occur, but the team should have a strong direction and controlled scope from the beginning.

*Outputs:* A document detailing the mission of the project, its key goals, the customers it seeks to serve, the use cases that will be enabled for those customers, and a road map showing the features that are planned to be built with a clear prioritization.

## Technology Selection

Once we have decided upon the scope of the project, we'll still need to make some technology choices. This phase will undoubtedly entail some experimentation in multiple directions, which could be parallelized among the Harvey Mudd team members. This phase will also require a great deal of collaboration with the Zendesk team members to understand what technologies would fit best in the Zendesk technology stack.

- What is the programming language... Javascript, Lua, a variant of an existing language, something new entirely?
- What will the execution environment be? If the programming language is Javascript, then `node.js` could be an excellent choice of execution environment.
- How will the technology scale, as it certainly will have to, to cope with increased demands for CPU, I/O, network bandwidth, and storage as customer usage increases?
- How will the technology communicate with the rest of Zendesk? Is it a microservice architecture? Does it use HTTP as transport, or some type of custom socket protocol? What do the message payloads look like: JSON, Thrift, Protocol Buffers, Avro, etc.?
- Will it be helpful to leverage some type of message bus for efficient communication such as Apache Kafka?
- How will the software be deployed? A good hypothesis is Docker containers that encapsulate the core engine in a way that can be easily deployed in a variety of environments.

*Outputs:* An architectural document with best initial guesses as to technology choices that will be employed, as well as architectural diagrams showing how the system works internally, and architectural diagrams showing how the entire system plugs into the larger Zendesk production landscape.

## Runtime Proof of Concept

This phase is to build a working proof of concept to demonstrate the viability and usefulness of the technology.

The runtime engine is a logical first piece to prototype, as it's the heart of the project and the key piece of code that enables the base functionality we are trying to achieve. Other pieces such as the user interface for the development experience are also important but not nearly as critical.

Once the POC is complete, the team should do a retrospective, take stock of the current state of the project and decide whether the plan needs to be adjusted before advancing down the same path.

The POC should be able to execute code and perform a simple integration with Zendesk. Performance and scalability should be considerations in the architecture but are not requirements. The developer experience doesn't need to be super-easy at this point... the most important thing is that something works to the point that the promise of the technology can be demonstrated.

*Outputs:* A working Proof Of Concept of the runtime technology with the ability to integrate with Zendesk.

## API Layer

Zendesk exposes most of its functionality via a rich public REST API. The Runtime POC will probably require customer code to use the REST API directly, or may supply a simple API wrapper that presents a more convenient OOP model but essentially wraps the REST API.

During the API Layer phase of the project, the team should study how ZenScript can most efficiently access the core functionality of Zendesk. Developer convenience, elegance of the resulting customer code and performance of the solution are important considerations.

Another important consideration is maintainability of the API layer. Can it be generated automatically in some way, so that it doesn't become just one more place to update when Zendesk API endpoints are updated or added?

*Outputs:* API layer, samples utilizing it, and supporting documentation.

## Security Sandbox

Zendesk customers will be excited by this project; unfortunately, so will hackers. Zendesk is a well-known company and is a target for hackers. They will attempt to suborn the system, trying to find back doors to escalate access to Zendesk and its partners, attempt to cause Zendesk outages, or even using ZenScript as a platform for hacking other sites. ZenScript needs to be resilient and secure in the face of these kinds of attacks.

These are just some example of what we must protect against:

- Attempts at excessive CPU consumption
- Attempts at excessive memory usage

- Attempts to do "internal reconnaissance" of the Zendesk production network
- Attempts to port-scan other servers, looking for targets
- Attempts to send spam, such as email or Web comments
- Attempts to cause Zendesk outages
- Attempts to create excessive amounts of data in the Zendesk system
- . . . and the list goes on

*Outputs:* Security sandbox layer in the Runtime Engine, along with supporting documentation explaining the security sandbox and the attacks it prevents.

## Container Deployment

Zendesk is in the process of migrating to a containerization-based infrastructure using Docker. Docker containers provide a straightforward mechanism for the Harvey Mudd team to package and deliver new versions of the runtime engine as well as other components in the project.

*Outputs:* One or more Docker images which can be easily deployed in Zendesk's test and production environments.

## Code Editing and Debugging Experience

An excellent suite of developer tools for ZenScript would probably be a Clinic program unto itself. However, it would be good to have an initial take on the ZenScript developer experience. Can we provide some simple means of viewing and editing code? Perhaps we get mileage out of integrating with popular source control platforms such as GitHub and Bitbucket.

Providing ZenScript developers with some ability to debug and troubleshoot their code is essential. This is probably about good logging behavior and quality error messages, but not sourcelevel debugging with breakpoints and expression evaluation like you might find in an IDE.

*Outputs:* An initial implementation or direction for a code editing and debugging experience for ZenScript.

## Document and Examples

No programming language is complete without documentation! The Harvey Mudd team won't be expected to produce final text that's ready to ship, but an important deliverable is an explanation of how the components of the system work, how the programming language is used, and a tutorial that gets new developers started with it, and a good set of examples.

*Outputs:* Documentation for the programming language and runtime system, tutorial, and example code.

# Development Practices

## Agile Development Methodology

Zendesk practices Agile development methodology, specifically Scrum. We encourage the Harvey Mudd team to adopt a lightweight Scrum process over the course of the project. Zendesk has found the system of a prioritized backlog, 24 week sprints, daily standup meetings, sprint planning, demos, and retrospective meetings to be enormously beneficial to the software development process. It also ensures consistency with the way Zendesk develops software, and is an excellent preparation for the way software is developed at many companies in the industry today.

## Peer Code Review

From the beginning, Zendesk Engineering has had a policy that all code commits are reviewed by a peer prior to being merged to master. We believe this is a helpful practice, as it usually results in better code and ensures developers are paying attention to each other's code and generalizing skills across the code base. Tools such as GitHub, Bitbucket and Gitlab streamline the process of code review even for distributed teams. We definitely encourage the Harvey Mudd team to practice peer code reviews on all commits, and this is also excellent preparation for the industry.

## Test Coverage

Zendesk believes that best practice for development is to include a suite of tests to exercise all code in the product. The decision to write tests should be made early, as it's difficult to retrofit into an existing code base. Automated tests are essential to being able to deploy and deliver software frequently. The use of a Continuous Integration system of some kind is also strongly encouraged.