

SSL をはじめよう

「なんとなく」から「ちゃんとわかる！」へ

mochikoAsTech 著

2020-03-01 版 **mochikoAsTech 発行**

はじめに

2020 年 3 月 mochikoAsTech

本書を手に取ってくださったあなた、こんにちは！ あるいは、はじめて。『SSL をはじめよう』の筆者、mochikoAsTech です。

SSL は好きですか？ それとも怖いですか？ 本書を書くまで、筆者は「ちゃんと分かっているとは言えないので、SSL を迂闊にさわるのはなんだか怖い」と感じていました。

SSL は、エンジニアのごく身近なところにあります。例えば「サイトをフル SSL 化する」「SSL 証明書を更新する」のような形で、誰しも一度は SSL と関わりをもったことがあるのではないでしょうか。しかし関わる機会が多い割に、「ちゃんと分かっているか」と言われるとちょっと自信がない、そんなエンジニアは筆者を含め、きっと一定数いると思っていました。

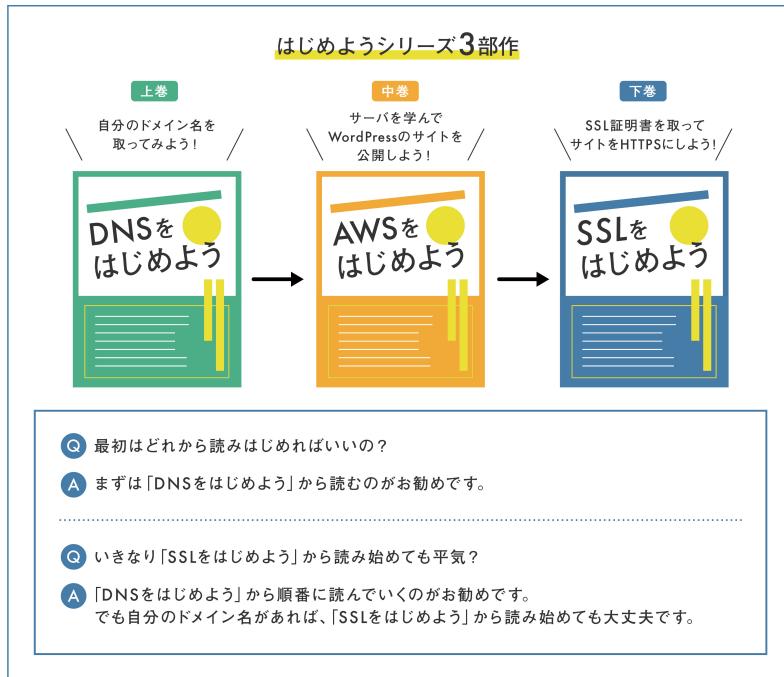
Google が 2014 年ごろからさかんに提唱している「HTTPS everywhere」（直訳すると「すべてを HTTPS で」）のもと、ウェブサイトの基準は、従来の「HTTP は普通、HTTPS にすれば安心」から、「HTTP は危険、HTTPS が普通」へと変わってきました。ますます SSL と関わる機会が増えてきて、心のどこかで「SSL、ちゃんと分かりたいなぁ…」と思っている。本書はそんな人のための一冊です。

理解度は「分かった！」「分かってなかった…」をくり返して、段々と上がっていくものです。まずは本書で、一緒に最初の「分かった！」まで進んでみましょう。

ちなみに本書「SSL をはじめよう」（以下 SSL 本）では、「DNS をはじめよう」（以下 DNS 本）で購入したドメイン名を使用します。DNS 本を読まずに SSL 本を読み進めていくと、第 2 章辺りで「ここで事前にあく抜きしておいた箇を取り出します」と言われて、「は？ あく抜きとかいつしてたの？！」という状態になります。「DNS は興味ないし面倒くさいんだけど…」という方も、できれば DNS 本を先にお読みいただいて、箇の下ごしらえ（＝ドメイン名の購入）を済ませた状態で SSL 本を開いてみてください。きっとその方が美味しくお召し上がりいただけます。なお SSL 本の第 1 章は、DNS 本を読んでいなくても問題ない内容ですので、とりあえずそのまま読み進めていただいても構いません。

またインフラやサーバに関する説明は、すでに「AWS をはじめよう」（以下 AWS 本）

で行なっていますので、本書では最低限にとどめています。本書で HTTPS のサイトを作ってみて、「インフラちょっと楽しいかも」と思われた方は、よかつたら後で AWS 本も召し上がってみてください。



▲図 1 はじめようシリーズ 3 部作

DNS 本、AWS 本、そして SSL 本の『はじめようシリーズ 3 部作』は、「サーバやインフラは怖いものではなくすごく楽しいものなんだよ」ということを、かつての私のようなインフラ初心者へ伝えたくて書いたシリーズです。

読んで試して「面白かった!」と思ってもらえたなら、そしてインフラを前より少しでも好きになってもらえたなら何より嬉しいです。

想定する読者層

本書は、こんな人に向けて書かれています。

- よく分からぬままネットの手順通りに SSL を設定している人
- 「サイトを HTTPS 化したいな」と思っている人

-
- ・証明書の購入や設置の流れがいまいち分かっていない人
 - ・SSL と TLS の関係性がよく分からぬ人
 - ・SSL 証明書が一体何を証明しているのか知らない人
 - ・これからシステムやプログラミングを学ぼうと思っている新人
 - ・ウェブ系で開発や運用をしているアプリケーションエンジニア
 - ・「インフラがよく分からぬこと」にコンプレックスのある人

マッチしない読者層

本書は、こんな人が読むと恐らく「not for me だった…（私向けじゃなかった）」となります。

- ・SSL/TLS の通信を C 言語で実装したい人
- ・「プロフェッショナル SSL/TLS」を読んで完全に理解できた人

本書の特徴

本書では実際にサーバを立てて SSL 証明書の設置を行い、HTTPS のサイトを作っています。手を動かして試してから仕組みを学べるので理解がしやすく、インフラ初心者でも安心して読み進められる内容です。Oracle Cloud の無料枠の中でサーバを立てて使用しますので、サーバ代はかかりません。SSL 証明書代のみ、1,000 円（税抜）かかります。

また SSL をめぐって実際にやってしまいがちな失敗、トラブルをとり上げて、

- ・こんな障害が起きたら原因はどう調べたらいいのか？
- ・問題をどう解決したらいいのか？
- ・どうしたら事前に避けられるのか？

を解説するとともに、クイズ形式で反復学習するためのドリルもついています。

本書のゴール

本書を読み終わると、あなたはこのような状態になっています。

- ・SSL 証明書がどんな役割を果たしているのか説明できる
- ・証明書を買うときの手順が分かっている
- ・意図せず「保護されていない通信」と表示されてしまったときの対処法が分かる

-
- 障害が起きたときに原因を調査できる
 - 読む前より SSL が好きになっている
 - SSL/TLS と併記されている「TLS」の意味が分かっている

免責事項

本書に記載されている内容は筆者の所属する組織の公式見解ではありません。

また本書はできるだけ正確を期すように努めましたが、筆者が内容を保証するものではありません。よって本書の記載内容に基づいて読者が行なった行為、及び読者が被った損害について筆者は何ら責任を負うものではありません。

不正確あるいは誤認と思われる箇所がありましたら、必要に応じて適宜改訂を行いますので GitHub の Issue や Pull request で筆者までお知らせいただけますと幸いです。

<https://github.com/mochikoAsTech/startSSL>

目次

はじめに	3
想定する読者層	4
マッチしない読者層	5
本書の特徴	5
本書のゴール	5
免責事項	6
第1章 SSL/TLS とは？	13
1.1 SSL/TLS ってなんですか？	14
1.1.1 SSL と TLS はどういう関係？	14
1.1.2 SSL イコール HTTPS ではない	14
1.2 「サイトを HTTPS 化する」とは何か？	15
1.3 どんなサイトでも必ず HTTPS にしなきゃだめ？	15
1.4 HTTP のままだと起きるデメリット	16
1.4.1 サイトが「保護されていない」と表示されてしまう	16
1.4.2 Wi-Fi スポットなどでセッションハイジャックされる恐れがある	17
1.4.3 相対的に検索順位が下がる	17
1.4.4 周りが HTTPS になるとリファラが取れなくなる	18
1.5 HTTPS 化すると得られるメリット	18
1.5.1 HTTP/2 との組み合わせで速度が向上するケースもある	18
1.5.2 SameSite の変更に対応できる	20
1.5.3 重要情報のアタリが付けにくくなる	20
第2章 Oracle Cloud のアカウントを作ろう	21
2.1 サイトを作るのにどうしてサーバがいるの？	22
2.2 サーバを立てるにはお金が必要？	23

2.3	なんで AWS じゃなくて Oracle のクラウドを使うの？	23
2.4	Oracle Cloud でアカウント登録	24
2.4.1	無料でアカウントを作ろう	24
2.4.2	〈トラブル〉どうしても SMS が届かない！ そんなときは？	30
2.4.3	パスワードと支払情報の登録	30
2.4.4	Oracle Cloud のコンソールにサインイン	35
第3章	サーバを立てて HTTP でサイトを見てみよう	39
3.1	事前準備	40
3.1.1	Windows でターミナル (RLogin) を準備する	40
3.1.2	Windows で SSH のキーペア (秘密鍵・公開鍵) を作成する	40
3.1.3	Mac でターミナルを準備する	44
3.1.4	Mac で SSH のキーペア (秘密鍵・公開鍵) を作成する	46
	【コラム】SSH の秘密鍵にパスフレーズは設定すべき？	47
3.2	コンピュートでサーバを立てる	48
3.2.1	OS は Oracle Linux 7.7 を使おう	49
3.2.2	作っておいた SSH の公開鍵を設置しよう	50
3.2.3	〈トラブル〉"Out of host capacity."が起きたらどうすればいい？	50
3.2.4	Always Free ではなく無償クレジットの枠でサーバを立てよう	52
3.2.5	サーバが起動するまで待とう	54
	【コラム】Oracle Cloud のコンピュートの金額計算方法	55
	【コラム】Oracle Cloud と AWS はどっちが安い？	56
3.2.6	接続先サーバの IP アドレス	56
3.3	サーバに SSH でログインしよう	57
3.3.1	Windows の RLogin を使ってサーバにログインしよう	57
3.3.2	〈トラブル〉サーバにログインできない！	63
3.3.3	Mac のターミナルを使ってサーバにログインしよう	64
3.4	NGINX をインストールしよう	66
3.5	Firewalld で HTTP と HTTPS を許可しよう	68
3.6	なぜか HTTP でサイトが見られない	69
3.6.1	サーバの手前にあるファイアウォールにも穴を空けよう	69
3.6.2	今度こそ HTTP でサイトを見てみよう	73
3.7	ドメイン名の設定をしよう	74
第4章	SSL 証明書を取って HTTPS でサイトを見よう	77

4.1	SSL 証明書にまつわる登場人物	78
4.2	秘密鍵 (startssl.key) を作ろう	79
	【コラム】SSL 証明書の秘密鍵にパスフレーズは設定すべき？	79
4.3	CSR (startssl.csr) を作ろう	81
	4.3.1 【ドリル】CSR で入力すべきなのは誰の情報？	83
	【コラム】openssl コマンドのユーティリティ（道具）たち	83
4.4	SSL 証明書の取得申請を出そう	85
4.5	DNS の設定をしよう	93
4.6	SSL 証明書と中間 CA 証明書をメールで受け取ろう	95
4.7	SSL 証明書をサーバに設置しよう	98
	4.7.1 Windows で SSL 証明書と中間 CA 証明書をアップロードしよう	98
	4.7.2 Mac で SSL 証明書と中間 CA 証明書をアップしよう	100
	4.7.3 証明書を 1 ファイル (startssl.crt) にまとめよう	100
4.8	NGINX で HTTPS のバーチャルホストを作ろう	102
4.9	HTTPS でサイトを開いてみよう	105
	【コラム】ロードバランサでも SSL ターミネーションできる	105
第 5 章	SSL/TLS の仕組み	107
5.1	SSL 証明書とは	108
	5.1.1 SSL サーバ証明書と SSL クライアント証明書	108
	5.1.2 SSL 証明書はどんなときに使われている？	109
5.2	SSL 証明書は異なる 3 つの仕事をしている	109
	5.2.1 HTTPS の実際の流れ	110
	5.2.2 正当性を証明する中間 CA 証明書	112
	5.2.3 ルート証明書はトラストストアにある	115
	5.2.4 ウェブページは 1 往復で表示されるわけじゃない	116
	5.2.5 HTTP の混在コンテンツはブロックされてしまう	117
	5.2.6 〈トラブル〉SSL 証明書の有効期限がうっかり切れてしまった .	119
5.3	SSL 証明書はどうしてあんなに値段に差があるの？	120
	5.3.1 同じ「SSL 証明書」でも 3 つの種類がある	120
	5.3.2 さよならグリーンバー	121
	5.3.3 任意のサブドメインで使えるワイルドカード証明書	123
	5.3.4 【ドリル】リダイレクトするだけでも www なしの証明書は必要？	123
	5.3.5 〈トラブル〉サイトを HTTPS 化したら古い端末で別サイトが表示された	124

目次

5.4 もっと SSL/TLS を学びたいときは	125
あとがき	127
PDF 版のダウンロード	128
Special Thanks:	128
レビュー	128
参考文献・ウェブサイト	128
著者紹介	131

第1章

SSL/TLS とは？

SSL ってなに？ TLS ってなに？ なんで HTTPS にしなきゃいけないの？
まずは素朴な疑問を解きほぐして、SSL をはじめる意味を見つけましょう。

1.1 SSL/TLS ってなんですか？

SSL (Secure Socket Layer) /TLS (Transport Layer Security) とは、**インターネット上で安全にデータを送受信するための約束事**（プロトコル¹）です。

SSL も TLS もあくまでプロトコル、つまり通信するための「約束事」なので、実際に通信するときは、そのプロトコルに従って実装されたソフトウェアを使います。SSL/TLS では、OpenSSL というオープンソースのソフトウェアを使うことがほとんど²です。

1.1.1 SSL と TLS はどういう関係？

SSL と TLS は別々の名前ですが、その役割に大きな違いはありません。「SSL3.0」からバージョンアップする際に、「SSL3.1」ではなく「TLS1.0」という新しい名前が付けられました。つまり「TLS は SSL の後継バージョン」ということです。

ちなみに名前がまだ SSL だったときの最後のバージョン「SSL3.0」は、重大な脆弱性³が見つかり、2015 年の RFC 7568⁴で、もう使わないよう「Do Not Use SSL Version 3.0」と示されています。

ですが SSL という言葉の認知度が高く、TLS と呼んでもピンとこない人の方が多いため、現状は分かりやすさと正確さを両立させて SSL/TLS と併記することが多いです。

本書では SSL/TLS のことを指して、SSL という言葉を使用します。

1.1.2 SSL イコール HTTPS ではない

サイト全体を「<https://>」から始まる URL にすることを、「常時 SSL 化」のように呼ぶため、皆さんの中には SSL = HTTPS だと思っている人がいるかも知れません。

HTTP と SSL を組み合わせて使うことで、通信が保護されるプロトコルが HTTPS (Hypertext Transfer Protocol Secure) です。ですが、SSL は HTTPS においてのみ使われるプロトコルではありません。例えばサーバにファイルをアップするときの FTP と

*¹ 例えば手紙は「メッセージを書いた便せんを封筒に入れて、表に宛先、裏に差出人を書き、重さに応じた切手を貼ってポストに入れる」という取り決めに従えば、きちんと相手に届きます。この取り決めのような、「インターネットで通信を行なう際、どんなデータをどんな方法で送受信するのか」という「約束事」のことをプロトコルと呼びます。SSL も TLS もプロトコルですし、HTTP や HTTPS、DNS も SMTP もプロトコルです

*² この後、第4章「SSL 証明書を取って HTTPS でサイトを見よう」で、実際に SSL 証明書を取得するときに使うコマンドも、openssl コマンドです

*³ 脆弱性（ぜいじやくせい）というのは悪用が可能なバグや設定不備のことです

*⁴ Deprecating Secure Sockets Layer Version 3.0 <https://tools.ietf.org/html/rfc7568>

SSL を組み合わせて使う FTPS (FTP over SSL) や、メール送信の SMTP と SSL を組み合わせて使う SMTPS (SMTP over SSL) など、HTTPS 以外にも SSL が使われている場面はたくさんあります。

繰り返しになりますが、SSL はインターネット上で安全にデータを送受信するためのプロトコルです。上位のアプリケーション層^{*5}が HTTP であれ、FTP であれ、**SSLを組み合わせて使うことで通信が暗号化により保護される**ようになるのです。

SSL = HTTPS ではない、ということを踏まえた上で、ここからは「サイトを HTTPS 化する」ことについて学んでいきましょう。

1.2 「サイトを HTTPS 化する」とは何か？

そもそもですが「サイトを HTTPS 化する」とは、なんでしょう？

本書では、「サイトの HTTPS 化」を、**ウェブサイト全体を「https://」から始まる URL にすること**と定義します。これは「常時 SSL 化」や「常時 SSL/TLS 化」、「フル SSL 化」、「AOSSL (Always On SSL の略)」といった言葉で表現されることもあります。

2014 年ごろはまだ、お問い合わせや会員登録など、個人情報を入力したり表示したりする一部のページのみを HTTPS にしているサイトが多く存在していました。ですが Google が HTTPS を強く推奨はじめたことで、「一部ページだけに限らず、ウェブサイト全体を HTTPS にしよう」という動きが段々と活発になってきました。

大手サイトが次々と HTTPS 化を始めたのが、2016 年から 2017 年ごろだったと記憶しています。例えばクックパッドは 2017 年 1 月^{*6}、日経新聞電子版は 2017 年 8 月^{*7}に、それぞれ HTTPS 化が完了したことを発表しています。

そして 2020 年現在、HTTPS 化の動きは、大手サイトだけでなくあらゆるサイトを対象にさらに進みつつあります。

1.3 どんなサイトでも必ず HTTPS にしなきゃだめ？

でも企業がやっているネットショップならまだしも、個人情報をやり取りするわけでもない、ただ個人の日記を公開しているだけのサイトでも、HTTPS 化はやらなければいけないのでしょうか？

確かにサイトを HTTPS 化すると「**通信が暗号化により保護される**」という大きなメ

^{*5} OSI 参照モデルのアプリケーション層のこと。エンジニア諸氏は誰しも、大学の授業や新卒研修で一度は「アセトナデブ」という語呂合わせを聞いたことがあるのでは…

^{*6} <https://techlife.cookpad.com/entry/2017/04/19/190901>

^{*7} <https://www.nikkei.com/topic/20170808.html>

リットがあります。ですが、2014年より前は「HTTPS化すれば通信が保護される。けれど特に保護すべきやりとりでなければ、HTTPS化しなくても悪いことが起きる訳ではない」という状況でした。

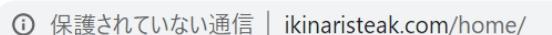
1.4 HTTP のままだと起きるデメリット

しかし2020年現在は、HTTPS化しないでいるとさまざまなデメリットが発生します。どんなデメリットが起きるのか、具体的に解説していきましょう。

1.4.1 サイトが「保護されていない」と表示されてしまう

GoogleはHTTPからHTTPSへの移行を強く推進しています。その施策の1つとして、Googleが提供するウェブブラウザ「Google Chrome」では、2018年7月にリリースされたバージョン68から、HTTPSでないページに対して「保護されていない通信」という表示をするようになりました。

Chromeでサイトを開いて、URLが「`http://`」で始まるものだった場合、次のようにURLの左側に「保護されていない通信」と表示（図1.1）されます。



① 保護されていない通信 | ikinaristeak.com/home/

▲図1.1 ChromeでHTTPのサイトを開くと「保護されていない通信」と表示される

サイトを見たとき、URLの真横に「保護されていない通信」と表示されたら、「なんだか危なさそう…見ない方がいいのかも…？」と心配になってしまいますよね。ChromeだけでなくFirefoxも、HTTPのサイトに対して錠前に赤い斜め線が入ったマークの表示（図1.2）を行っており、サイトをHTTPS化しないことで、訪れたユーザにサイトが「安全でない」と思われるてしまう状況にあります。



▲図1.2 FirefoxでHTTPのサイトを開くと、錠前に赤い斜め線が入ったマークが表示される

1.4.2 Wi-Fi スポットなどでセッションハイジャックされる恐れがある

続いての「HTTP のままだと起きるデメリット」はセッションハイジャックです。

例えば、買い物をしようと思って楽天のサイトを開くと、今日はまだログインしていないのに、ログイン済みのページが表示されることがあります。これは以前ログインした際に、サイトから Cookie で渡された「セッション ID」（一時的な通行証のようなもの）をブラウザが保持していて、再びサイトを開いたときに提示することで、ログイン済みのユーザとして扱われるからです。

ログインページだけが HTTPS になっているサイトだと、それ以外のページを HTTP で開いたとき、Cookie に Secure 属性が付いていなければ、この「セッション ID」は暗号化されない状態で送信されてしまいます。では、街中の誰でも使える Wi-Fi スポットに、スマホやタブレットを繋いだ状態で、HTTP のページを開いて「セッション ID」が送られたらどうなるでしょう？

なんと同じ Wi-Fi につないでいる、悪意の第三者によって、暗号化されていない「セッション ID」を盗まれ、なりすましてサイトにログインされる恐れがあります。これが「セッションハイジャック」と呼ばれる攻撃です。

こうした攻撃を防ぐには、サイト全体を HTTPS にして、常に Cookie の「セッション ID」を保護しておく必要があります。またログインなどが一切ない、公開情報しか載せていないサイトであっても、Wi-Fi スポットなどで見知らぬ誰かに「あの人はどんなサイトを見ているのだろう？」とデータを窺視されるリスクもあります。

例えば「妊活情報のブログを長時間読んでいた」「特定の病気について調べていた」など、どんなサイトを見ていたのか？ という情報の中にも、人に知られたくないことはたくさんあります。サイトとの通信が HTTPS で保護されていれば、このような情報を盗み見られるリスクを低減できます。

1.4.3 相対的に検索順位が下がる

さらに、インターネット全体で HTTPS 化が進むことによって、HTTP のサイトに起きるデメリットもあります。

Google は「HTTPS everywhere」、つまり「(お問い合わせや会員登録といった一部のページに限らず) すべてを HTTPS で！」を提唱しており、2014 年 8 月の時点で既に、HTTPS に対応しているウェブサイトを検索ランキングで優遇する方針も発表^{*8}してい

^{*8} Google ウェブマスター向け公式ブログ [JA]: HTTPS をランキング シグナルに使用します <https://webmaster-jp.googleblog.com/2014/08/https-as-ranking-signal.html>

ます。

Google のランキングアルゴリズムでは、「サイトが HTTPS 化されているか」が指標のひとつとなっています。もちろんたくさんある指標の中のひとつで、他の指標ほどウェイトは大きくない、とされていますが、競合サイトの HTTPS 化が進む中、自分のサイトだけ HTTP のままでいると、相対的に検索順位が下がる可能性があります。

1.4.4 周りが HTTPS になるとリファラが取れなくなる

こちらは Google アナリティクスなどを使って、サイト流入元の情報を確認している方にとって、重要と思われるデメリットです。

自社のサイトが HTTP だと、HTTPS の他社サイトからリンクを踏んで飛んできた場合に、リファラ（利用者が直前に訪問していたサイトの情報）を取得することができません。HTTP のサイトで Google アナリティクスを使っている方は、実際に Google アナリティクスのページを開いて、集客の「参照元/メディア」を確認してみてください。アクセス元が「(direct) / (none)」と表示されて、どこから飛んできたのか分からぬものがありますか？ その中には、ブラウザのブックマークや、メール内のリンクから飛んできた、本当に「直前に訪問していたサイトが存在しないもの」だけでなく、HTTPS のサイトから飛んできたアクセスも含まれています。

今後、周囲のサイトの HTTPS 化が進んで、自社のサイトだけが HTTP で取り残されると、この「リファラが取得できる割合」はますます下がっていくことになります。自社のサイトを HTTPS にすれば、他社の HTTPS サイトから飛んできた場合でも、リファラを取得できるようになります。

1.5 HTTPS 化すると得られるメリット

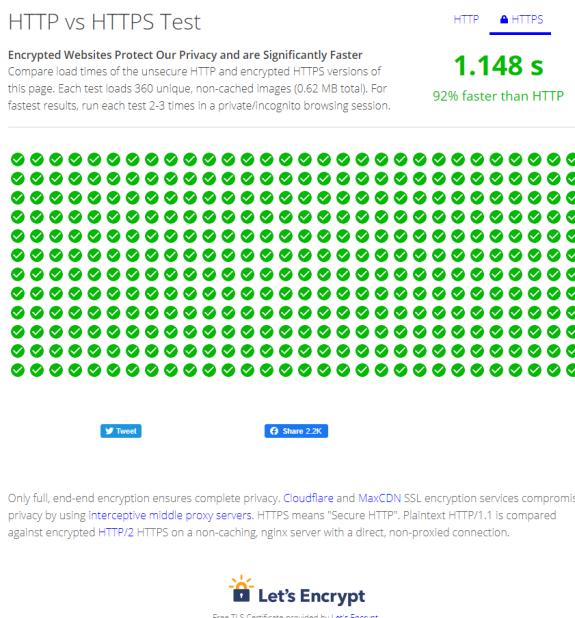
HTTP のままでいると起きるデメリットだけでなく、HTTPS 化すると得られるメリットももちろんあります。

1.5.1 HTTP/2との組み合わせで速度が向上するケースもある

かつては、HTTPS にすると、暗号化と復号のオーバーヘッドで HTTP のときよりもサイトの表示が遅くなる、というのが常識でした。しかし、サーバ側、クライアント側とともに CPU の性能が向上したことで、2020 年現在、今はオーバーヘッドは重要視するほ

どではなくなっています。^{*9}

また HTTPS にすることで、HTTP/2 という HTTP プロトコルの次期バージョンを利用できる^{*10}ようになります。SSL のオーバーヘッドは特にハンドシェイクに集中しているため、長期にわたって接続したまで、リクエストとレスポンスを多重化できる HTTP/2 と組み合わせると、ハンドシェイクの回数が減る分だけ接続効率が上がります。ページの作りによってもちろん向き不向きがあり、HTTPS にしたことで表示速度が落ちるケースもあると思います。ですが、例えばサイズが小さな画像を大量に表示するようなページ（図 1.3）^{*11}は、HTTP/2 の恩恵を受けやすく、速度の向上が期待できます。



▲図 1.3 HTTP と HTTPS の速度比較テストをするサイト

さらに TLS1.3になると、TLS1.2と比べてハンドシェイクに要するやりとりの回数や所要時間が大幅に短くなり、より一層の速度改善が見込まれます。

^{*9} サーバの手前に配置して、暗号化や復号だけを行なう専用機器の「SSL アクセラレータ」を導入する話も、最近はほぼ聞かなくなりました

^{*10} HTTP/2 は、仕様上は HTTP にも対応していますが、ブラウザ側が HTTPS でしか HTTP/2 を利用できないようになっているため、実質的には HTTP/2 を使う際は HTTPS が必須となります

^{*11} HTTP vs HTTPS Test <https://www.httpvshttps.com/>

1.5.2 SameSite の変更に対応できる

2020年2月にリリースされたChromeのバージョン80^{*12}から、CookieのSameSiteの設定が従来より厳しくなり、今まで設定なしで使えていたクロスサイトCookieが、デフォルトでは使えないように順次変更されていくことが発表されました。

ですが「SameSite=None; Secure」の設定をすれば、クロスサイトCookieは引き続き利用できます^{*13}。そしてSecure属性を追加するには、HTTPSでの接続が必須です。

このように、HTTPSであればSameSiteの変更にも対応できる、というメリットがあります。

1.5.3 重要情報のアタリが付けにくくなる

最後に、これは自分のサイトだけでなく、インターネット上のサイト全体がHTTPSに対応したときに得られるメリットです。

流れしていくデータのほとんどが平文な中で、たまに暗号化されたデータがやってくると、データを窺視していた悪意の第三者は「暗号化されているということは、あれは重要な情報だな！」とアタリを付けて狙うことができます。ですが、すべてのサイトがHTTPSになって、流れてくるデータがすべて暗号化されるようになれば、どれが重要なデータなのか、というアタリが付けにくくなります。

普段着の人が多いところに、高そうなスーツを着てアタッシュケースと手首を手錠で繋いでいる人がポツンと立っていたら目立ちますし、明らかに「大事なものを持っているんだな」と分かりますよね。でもみんなが同じスーツを着てアタッシュケースを持てば、それが本当に大事な情報なのか、ぱっと見では分からなくなります。木は森に隠せ、という戦法です。

このように、HTTPのままだと起きるデメリットと、HTTPSにすることで得られるメリットの両方があるので、**どんなサイトもHTTPS化する意味がある**ということですね。

^{*12} <https://developers-jp.blogspot.com/2019/11/cookie-samesitenone-secure.html>,
<https://developers-jp.blogspot.com/2020/02/2020-2-samesite-cookie.html>

^{*13} 2020年2月時点ではこの方法で回避できますが、将来的には制約がより厳しくなる可能性ももちろんあります

第2章

Oracle Cloud のアカウントを作 ろう

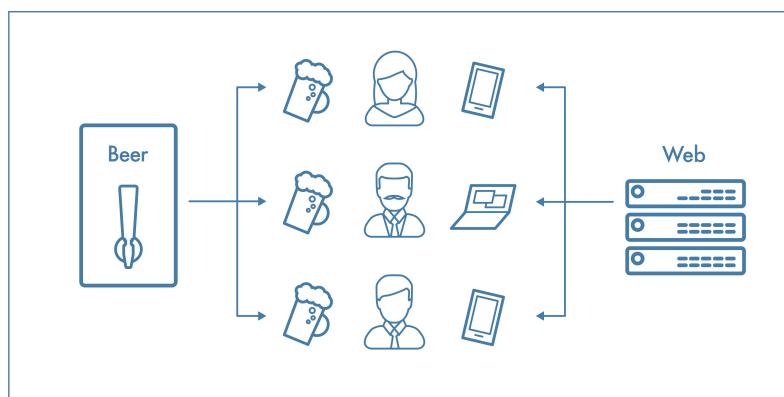
この章では Oracle Cloud というクラウドでアカウントを作ります。
SSL を理解するには、実際に手を動かしてやってみるのがいちばんです。
その下準備としてまずはアカウント作成からはじめましょう。

2.1 サイトを作るのにどうしてサーバがいるの？

HTTPS のサイトを作るのに、必要な材料は次の 3 つです。

- ウェブサーバ
- ドメイン名
- SSL 証明書

ところで、どうしてサイトを作りたいだけなのに、ウェブサーバが必要なのでしょう？そもそもですが、サーバとは**クライアントに対してサービスを提供するものです**。居酒屋にあるビアサーバに「ビールをください」というリクエストを投げる…つまりハンドルを「開」の方へひねると、ビールというレスポンスが返ってきます。同様にあなたがブラウザで URL を入力したり、リンクをクリックしたりして、ウェブサーバに対して「ウェブページを見せてください」というリクエストを投げたら、ウェブページというレスポンスが返ってきます。（図 2.1）



▲図 2.1 クライアントに対してサービスを提供するのがサーバ

つまり、せっかく HTML や画像でウェブサイトのコンテンツを作っても、そのコンテンツを提供するウェブサーバがなければ、サイトはあなたのパソコンの中しか見られず、インターネットで公開できないのです。^{*1}

^{*1} サーバについては、はじめようシリーズの 2 作目、「AWS をはじめよう」の「CHAPTER1 インフラとサーバってなに？」で、より詳しく解説しています。仮想サーバと物理サーバ、クラウドとオンプレミス、ホストサーバとゲストサーバなどサーバ周りの用語をもう少し理解したい！ という方はそちらも併せて読んでみてください

というわけでウェブサイトをインターネットで公開するために、まずはウェブサーバを立てましょう。

2.2 サーバを立てるにはお金が必要？

ウェブサイトを公開するにはサーバが必要です。そしてサーバを立てるには、普通はお金がかかります。ですがオラクルが提供している「Oracle Cloud（オラクル クラウド）」というサービスには、なんと有効期限なしでずっと無料で使えるAlways Freeという枠があります。Always Free の範囲^{*2}内であれば、サーバも無料で立てられます。

さらに Always Free とは別に、アカウント作成から 30日間だけ有効な300ドル分の無償クレジットも付いてきますので、Always Free の範囲外のサービスはそちらで試せます。本書ではこの Always Free と、無償クレジットの範囲内で Oracle Cloud を利用していきます。

Oracle Cloud は名前のとおりオラクルが提供しているクラウドです。あなたが「ウェブサイトを公開したいなあ…だからサーバが必要だ！」と思ったとき、従量課金で、すぐに使えて、性能や台数の増減も簡単にできるのがクラウドです。Oracle Cloud なら、ブラウザでぼちぼちとスペックを選んでいくだけで、すぐにサーバが立てられます。

2.3 なんで AWS じゃなくて Oracle のクラウドを使うの？

クラウドは Oracle Cloud だけではありません。かの有名な AWS こと Amazon Web Services や、Google の Google Cloud Platform^{*3}、Microsoft の Azure^{*4}、その他にも国内クラウドとしてさくらインターネットがやっているさくらのクラウド^{*5}、お名前.com でお馴染み GMO グループの GMO クラウド^{*6}などなど、たくさんのクラウドが存在しています。

2019年11月時点、クラウド市場では AWS がシェア約40%でトップを独走中^{*7}です。そのため仕事で AWS を使ったことがある、あるいはこれから使う予定だ、というエンジニアも多いと思います。

^{*2} Always Free の範囲については、<https://www.oracle.com/jp/cloud/free/> を確認してください

^{*3} <https://cloud.google.com/>

^{*4} <https://azure.microsoft.com/ja-jp/>

^{*5} <https://cloud.sakura.ad.jp/>

^{*6} <https://www.gmocloud.com/>

^{*7} IaaS + PaaS クラウド市場、AWS の首位ゆるがず。AWS、Azure、Google、Alibaba の上位4社で市場の7割超。2019年第3四半期、Synergy Research Group — Publickey https://www.publickey1.jp/blog/19/iaaspasawsazuregooglealibaba4720193synergy_research_group.html

しかし最近は、Alibaba Cloud や Oracle Cloud といった後発のクラウド事業者も追い上げを見せています。こうした新興のクラウドは、先に行く AWS を見て学んだ上で生まれてきただけあって、よりスマートな作りになっているのがいいところです。

本来であれば、クラウドを選定する際の基準は、そのクラウド上で動かすサービスの性質によって異なるはずです。あなたが動かしたいサービスには、一体どのクラウドが適しているのでしょうか？

本書では次の2つを選定基準として、それに適した Oracle Cloud で学びを進めていきたいと思います。

- SSL 証明書を自分で取得して設置する一通りの流れを試したい
- できるだけお金をかけずに無料で試したい

2.4 Oracle Cloud でアカウント登録

まずは Oracle Cloud のアカウントを作ります。お手元に次の2つを用意してください。

- クレジットカード
- SMS^{*8}受信が可能な携帯電話（電話番号認証で使用するため）

なお本書では、前述のとおり Always Free と 300 ドル分の無償クレジットという無料枠の範囲内で、Oracle Cloud を利用していきます。クレジットカードの登録は主に本人確認のために行なうもので、無料アカウントから有償アカウントへ手動でアップグレードしない限り、勝手に課金はされないので安心してください。

2.4.1 無料でアカウントを作ろう

「Oracle Cloud 無料」で検索（図2.2）したら、いちばん上の [Oracle Cloud Free Tier | Oracle 日本]^{*9}をクリックします。

^{*8} ショートメッセージサービスの略。宛先に電話番号を指定してメッセージを送れるサービス

^{*9} <https://www.oracle.com/jp/cloud/free/>

2.4 Oracle Cloud でアカウント登録



▲図 2.2 「Oracle Cloud 無料」で検索

Oracle Cloud Free Tier^{*10}のページが表示されたら、[今すぐ始める（無償）] をクリックします。（図 2.3）



▲図 2.3 [今すぐ始める（無償）] をクリック

[Oracle Cloud へのサインアップ] と表示されたら、[電子メール・アドレス] と [国/

^{*10} ずっと無料の AlwaysFree と、30 日間だけ有効な 300 ドル分の無償クレジット、この 2 つを総称して「Free Tier（無償ティア）」と呼ぶようです

第2章 Oracle Cloud のアカウントを作ろう

地域】を入力して、使用条件を確認した上で【次】をクリックします。(図 2.4)
後で分からなくなないように、登録した内容を表 2.1 にメモしておきましょう。

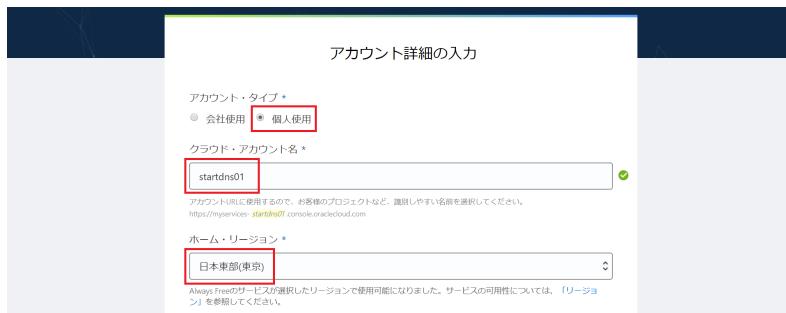
▼表 2.1 Oracle Cloud に登録した情報

項目	例	あなたが登録した情報
電子メール・アドレス	startdns.01@gmail.com	
国/地域	日本	



▲図 2.4 入力したら【次】をクリック

次は【アカウント詳細の入力】です。今回は仕事ではなく個人での利用ですので【アカウント・タイプ】は【個人使用】を選択してください。【クラウド・アカウント名】には**任意のアカウント名**を入力します。【クラウド・アカウント名】には英字小文字と数字のみ使えます。記号や英字大文字は使えないで注意してください。筆者は startdns01 にしました。(図 2.5)



▲図 2.5 [クラウド・アカウント名] には任意の名前を入力

[ホーム・リージョン] は [日本東部(東京)] を選択します。Oracle Cloud は世界の各地域にデータセンターを所有しており、サーバはそのデータセンターの中で元気に動いています。この [ホーム・リージョン] とは、**どの地域のデータセンターを使うか？を指定するものです。**

ウェブサイトにアクセスするとき、パソコンのある場所からサーバまで物理的に距離が遠いと、それだけ通信にも時間がかかるて応答時間も遅くなります。日本国内向けにウェブサイトを開設する場合は、基本的にこの [日本東部(東京)] のリージョンを選びましょう。[ホーム・リージョン] で [日本東部(東京)] を選択していても、他のリージョンも利用できますので安心してください。(表 2.2)

▼表 2.2 Oracle Cloud に登録した情報

項目	例	あなたが登録した情報
アカウント・タイプ	個人使用	
クラウド・アカウント名	startdns01	
ホーム・リージョン	日本東部(東京)	

この [ホーム・リージョン] は後から変更ができません。そして Always Free は、選択した [ホーム・リージョン] においてのみ利用可能です。

2020 年 2 月現在、Oracle Cloud の [日本東部(東京)] のリージョンは人気が高く、Always Free 向けのリソースが不足していてサーバが立てられない、という事態がしばしば起きています。[米国東部(アッシュバーン)] をホーム・リージョンにすれば、このリソース不足は回避できますので、Always Free を確実に使いたい方は、[ホーム・リージョン] で [米国東部(アッシュバーン)] を選択してください。本書では Always Free 向けのリソースが不足していた場合は、無償クレジットを使用してサーバを立てますので、[日本東部(東京)] を選択して問題ありません。

第2章 Oracle Cloud のアカウントを作ろう

続いて名前や住所を入力していきます。入力内容は日本語表記で構いません。個人利用なのですが「部門名」が必須であるため、ここでは「個人」と入力しておきましょう。「名」、「姓」、「部門名」、「住所」、「市区町村」、「都道府県」、「郵便番号」のすべてを入力してください。(図 2.6)

The screenshot shows a registration form with various input fields. The fields highlighted with red boxes are: '名' (Name) containing '猫村', '姓' (Last Name) containing 'もち子', '部門名' (Department Name) containing '個人', '住所' (Address) containing '新宿四丁目1番6号' and 'JR新宿ミライナタワー', '市区町村' (City/Town/Village) containing '新宿区', '都道府県' (Prefecture) containing 'TOKYO', and '郵便番号' (Postal Code) containing '160-0022'. Other visible fields include '役職' (Position), '国/地域' (Country/Region) set to '日本', and a search icon next to the prefecture field.

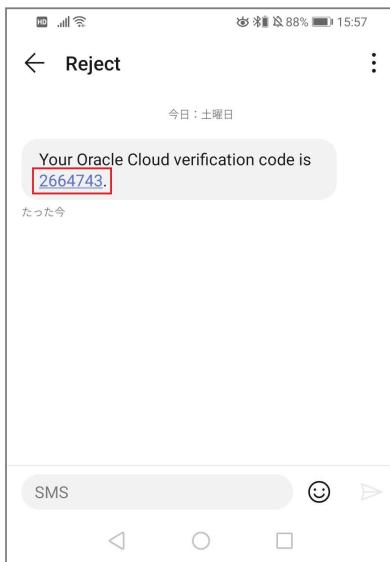
▲図 2.6 名前や住所を入力

では最後に「モバイル番号」です。国番号は「日本 (81)」を選択して、自分の携帯電話番号を入力します。このとき電話番号の先頭の 0 は不要です。例えば「090-〇〇〇〇-〇〇〇〇」という携帯電話番号であれば「90-〇〇〇〇-〇〇〇〇」と入力してください。携帯電話番号を入力したら「次: モバイル番号の確認」をクリックします。(図 2.7)

The screenshot shows a step titled 'モバイル番号' (Mobile Number). It displays a dropdown menu for '日本 (81)' and a text input field containing '90'. Below the input field is a note about verification codes being sent via email to 'startdns.0@gmail.com'. A large blue button at the bottom is labeled '次: モバイル番号の確認' (Next: Mobile number confirmation).

▲図 2.7 携帯電話の番号を入力

数分以内に「Your Oracle Cloud verification code is 〇〇〇〇〇〇〇〇.」と書かれたSMSが、入力した携帯電話番号宛てに届きます。(図 2.8)



▲図 2.8 コード（7 行の数字）が SMS で届いた

SMS で届いた 7 行の数字を [コード] に入力して、[コードの確認] をクリックします。
(図 2.9)



▲図 2.9 SMS で届いた数字を [コード] に入力して [コードの確認] をクリック

2.4.2 <トラブル> どうしても SMS が届かない！ そんなときは？

携帯電話番号を入力したのに SMS が届かないときは、まず自分が契約している携帯キャリアの迷惑メール設定で、SMS をスパムとしてはじく設定をしていないか確認してみましょう。例えば海外の事業者から送信された SMS を拒否する設定になっていたり、海外からの着信を拒否する設定になっていると、SMS が届かないことがあるようです。^{*11}

ちなみに筆者の場合は、特に設定変更をせず同じ番号で 2 回試してみたのですが、最初は届かず、2 回目でようやく届きました。

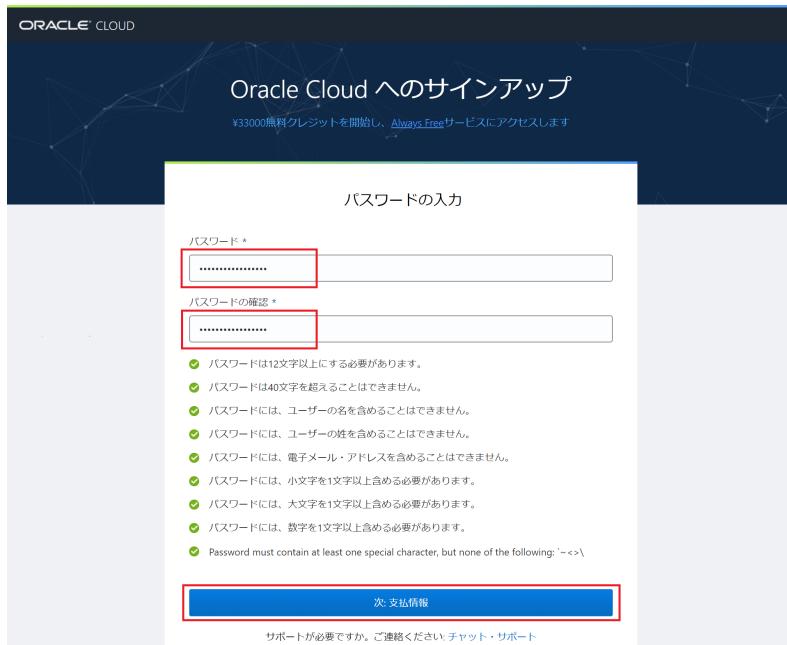
迷惑メールの設定を確認した上で何回か試して、それでも SMS が届かなかったら、ページ下部の【サポートが必要ですか。ご連絡ください: チャット・サポート】からサポートにチャットで問い合わせてみましょう。残念ながら英語でしか対応してもらえませんが、"I am trying to register on Oracle Cloud. But I can't receive SMS. What should I do?"（アカウント登録しようとしてるけど SMS が届かないの、どうしたらいい？）という感じで聞いてみましょう。すぐに「じゃあ登録情報をこのチャットで教えて。そうしたらこちらでコードを発行して、このチャットで伝えてあげる」（意訳）とサポートしてもらえます。

2.4.3 パスワードと支払情報の登録

正しいコードを入力すると、【パスワードの入力】が表示されます。【パスワード】と【パスワードの確認】を入力して、【次: 支払情報】をクリックします。（図 2.10）

^{*11} 「Oracle Cloud の SMS は海外の事業者から届く」という確証がある訳ではないです。あくまで SMS が届かないときによくある話だと思ってください

2.4 Oracle Cloud でアカウント登録



▲図 2.10 パスワードを入力して [次: 支払情報] をクリック

パスワードを入力すると、今度は「支払情報」のページが表示されます。

繰り返しお伝えしているとおり、Oracle Cloud には AlwaysFree と無償クレジットという無料枠があります。本書ではこの無料枠の範囲内で Oracle Cloud を使っていきますが、それでもクレジットカードは登録しておく必要があります。

なお「支払情報」のページに記載されているとおり、この後、管理画面で「アカウントのアップグレード」という作業をして、有償アカウントへ切り替えない限り、請求は絶対に発生しません^{*12}ので安心してカード情報を登録してください。

[クレジット・カード詳細の追加] をクリック（図 2.11）します。

^{*12} 300 ドル分の無償クレジットを使い切るか、あるいはアカウントを作成してから 30 日が経過すると、メールでお知らせが届きます。そこからさらに 30 日の間に有償アカウントへアップグレードしなければ、Always Free のサービスを除いて、無償クレジットで作ったサーバやデータは削除されます。AWS のように、サーバを削除し忘れたまま無料期間が終わって、うっかり課金されてしまった…という事態は、Oracle Cloud では起きないので安心してください

第2章 Oracle Cloud のアカウントを作ろう



▲図 2.11 [クレジット・カード詳細の追加] をクリック

【ご注文者様情報】はそのまま変更不要です。【カード情報】の【カードの種類】を選択し、【カードの番号】、【有効期限】、【CVN】を入力したら【Finish】をクリックします。(図 2.12)*¹³

A screenshot of the 'Card Information' input form. It includes fields for 'Card Type' (radio buttons for Visa, Mastercard, Amex, and JCB), 'Card Number' (input field), 'Expiration Date' (dropdown menus for month and year), and 'CVN' (input field). A note below the CVN field states: 'このコードは、クレジットカードの裏面または表面に印字されている3桁または4桁の番号です。' (This code is a 3 or 4 digit number printed on the back or front of the credit card). A green 'Finish' button is at the bottom right of the form.

▲図 2.12 カード情報を入力して [Finish]

【クレジット・カード詳細をご提供いただきありがとうございます。】と表示（図 2.13）

*¹³ Oracle Cloud では、クレジットカード登録時に「1 ドル認証」と呼ばれる認証方法で、そのクレジットカードが決済可能かをチェックしています。実際に請求は行なわれないのですが、クレジットカードによってはこの 1 ドル認証を不審な決済と判断して通さないため、それによってエラーが発生することがあります。その場合は別のクレジットカードで試すか、Oracle Cloud のチャット・サポートで問い合わせてみてください

2.4 Oracle Cloud でアカウント登録

されたら、支払い情報の登録は完了です。Oracle Cloud の Service Agreement^{*14}を確認した上で、チェックボックスにチェックを入れて、「[サインアップの完了]」をクリックします。



▲図 2.13 チェックを入れて [サインアップの完了] をクリック

これで Oracle Cloud でのアカウント作成の手続きはおしまいです。[アカウントの設定が完了するまでお待ちください。] と表示（図 2.14）されます。準備が整うとサインイン画面にリダイレクトされますが、この [アカウントの設定が完了するまでお待ちください。] の画面でかなり時間がかかるので、一度ブラウザを閉じてしまって構いません。頑張った自分を褒めてあげて、一旦休憩にしましょう。

*14 <https://www.oracle.com/goto/oraclecsa-jp-en>

第2章 Oracle Cloud のアカウントを作ろう

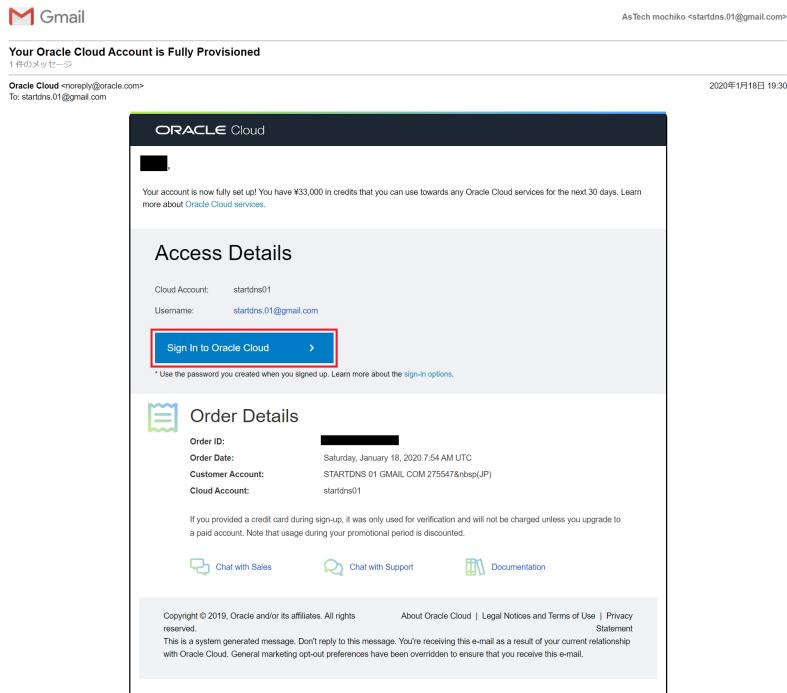


▲図 2.14 アカウント作成の手続きはおしまい

数時間後^{*15}、[Your Oracle Cloud Account is Fully Provisioned] という件名で、準備完了を知らせるメールが届きます。メール本文の [Sign In to Oracle Cloud] をクリックしましょう。(図 2.15)

^{*15} 筆者の場合は、メールが届くまで約 2 時間半かかりました

2.4 Oracle Cloud でアカウント登録



▲図 2.15 数時間後、準備完了を知らせるメールが届く

2.4.4 Oracle Cloud のコンソールにサインイン

メールの [Sign In to Oracle Cloud] をクリックすると、コンソールへのサインイン^{*16}画面が表示されます。(図 2.16)

[ユーザー名] には先ほど登録したメールアドレスを入力します。^{*17} [パスワード] を入力して、[サイン・イン] をクリックしてください。

*16 日本語だとログインの方が馴染みがあるかも知れませんが、サインインはログインと同じ意味です

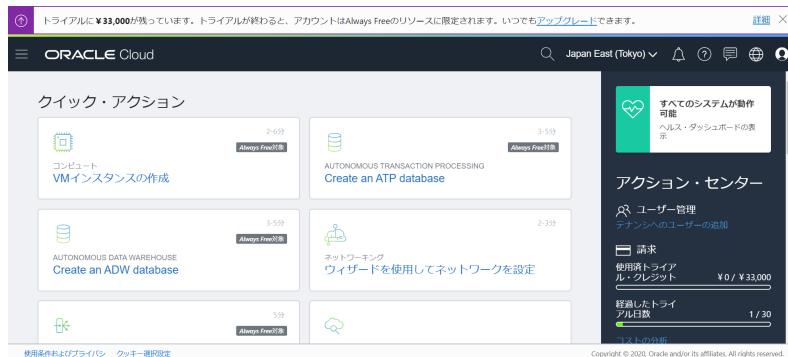
*17 メールにも書いてありますが、ここでの [ユーザー名] とは [クラウド・アカウント名] (筆者の場合は startdns01) ではなく、[メールアドレス] のことです。紛らわしいのでご注意ください

第2章 Oracle Cloud のアカウントを作ろう



▲図 2.16 [ユーザー名] と [パスワード] を入力して [サイン・イン]

おめでとうございます！ Oracle Cloud の管理画面、コンソールにサインインできました。



▲図 2.17 コンソールにサインインできた！

なお今後、コンソールにサインインしたくなったら、いちいち Oracle Cloud からのメールを探してリンクを踏む必要はありません。まずは Oracle のトップページ^{*18}を開いて、右上の人マークをクリックします。そして「クラウドにサインイン」をクリックしてください。

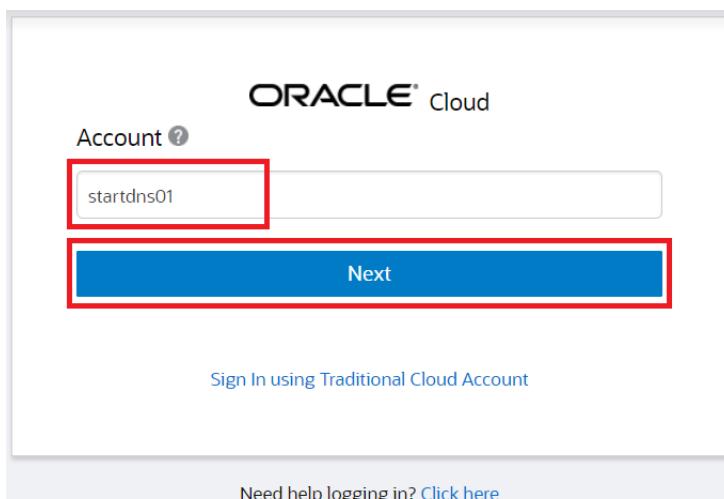
*18 <https://www.oracle.com/jp/>

2.4 Oracle Cloud でアカウント登録



▲図 2.18 右上的人物マークをクリックして [クラウドにサインイン]

サインインのページ^{*19}で [Account] の欄にクラウド・アカウント名^{*20}を入力して [Next] をクリックすれば、メールのリンクを踏んだときと同じ [サイン・イン] のページにたどり着けます。あとは同じように [ユーザー名] にはメールアドレスを、[パスワード] にはパスワードを入力して、[サイン・イン] をクリックするだけです。



▲図 2.19 [Account] の欄にクラウド・アカウント名を入力して [Next] をクリック

*19 <https://www.oracle.com/cloud/sign-in.html>

*20 筆者の場合は startdns01 です。アカウント登録時に、あなたの [クラウド・アカウント名] をメモしているはずです。P26 に戻って確認してみましょう

第3章

サーバを立てて HTTP でサイトを見てみよう

HTTPS のサイトを作るのに、必要な材料は次の 3 つです。

- ウェブサーバ
- ドメイン名
- SSL 証明書

まずはウェブサーバとドメイン名を用意して、HTTP でサイトを見てみましょう。

3.1 事前準備

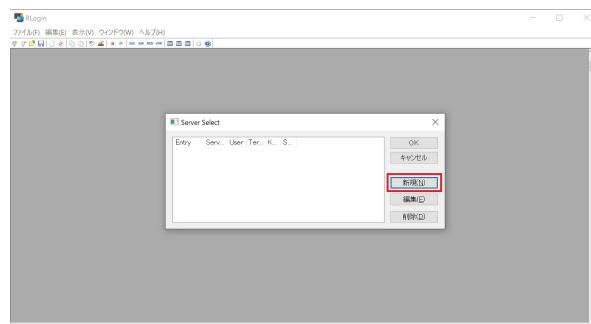
まずは、サーバを立てるときに必要なターミナルソフトを準備して、キーペアの作成を行ないます。Windows、Mac の順番で手順を説明します。

3.1.1 Windows でターミナル（RLogin）を準備する

Windows のパソコンを使っている方は、サーバを立てる前に「ターミナル」と呼ばれる黒い画面のソフトをインストールしておきましょう。サーバにログインするときには、このターミナルを使います。ターミナルのソフトには色々な種類がありますが、本書では RLogin^{*1}を使用します。RLogin のインストール方法は「AWS をはじめよう」^{*2}で説明していますので、本書では省略します。インストールした RLogin のフォルダは、デスクトップにあるものとします。

3.1.2 Windows で SSH のキーペア（秘密鍵・公開鍵）を作成する

それでは RLogin を使ってキーペアを作成します。起動した RLogin で【新規 (N)】をクリックしてください。



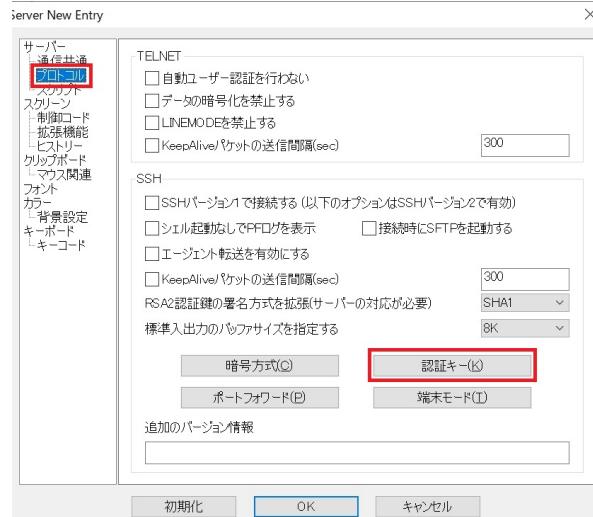
▲図 3.1 【新規 (N)】をクリック

左メニューの【プロトコル】を選択して、【認証キー (K)】をクリックします。

*1 <http://nanno.dip.jp/softlib/man/rlogin/>

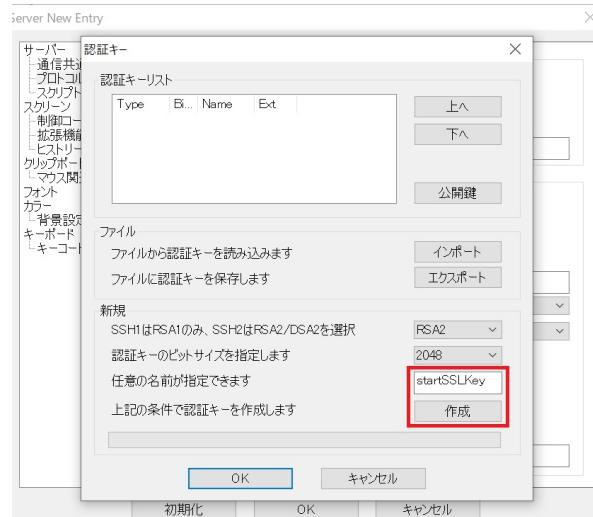
*2 <https://mochikoastech.booth.pm/items/1032590>

3.1 事前準備



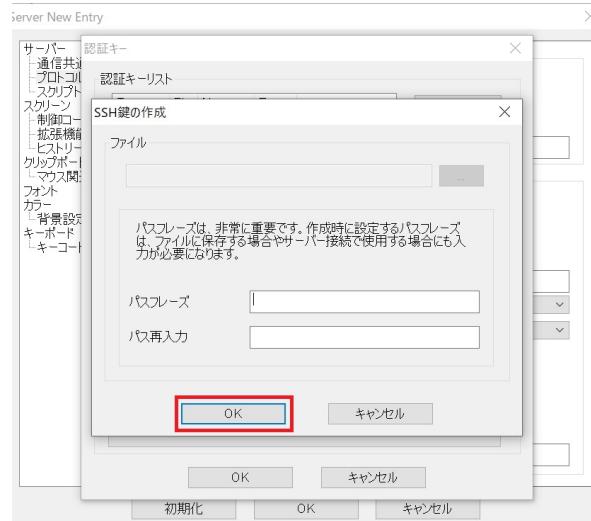
▲図 3.2 [プロトコル] を選択して [認証キー (K)] をクリック

[任意の名前が指定できます] に [startSSLKey] を入力して、[作成] をクリックします。



▲図 3.3 [startSSLKey] を入力して [作成] をクリック

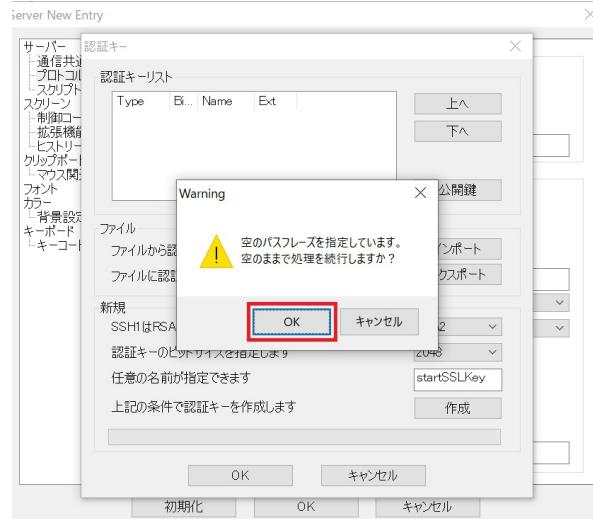
[パスフレーズ] と [パス再入力] には何も入力せず、[OK] をクリックします。^{*3}



▲図 3.4 何も入力せず [OK] をクリック

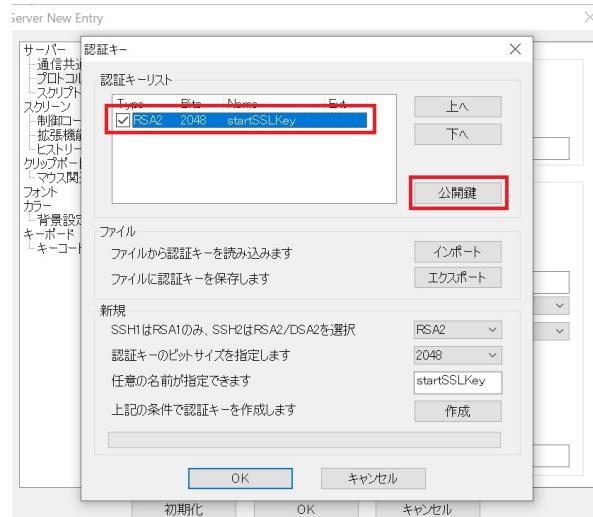
[空のパスフレーズを指定しています。空のままで処理を続行しますか?] と表示されますが、そのまま [OK] をクリックします。

^{*3} このパスフレーズは秘密鍵の保護に用いられます。パスフレーズを設定しなくてよいのか、についてはコラムで後述します



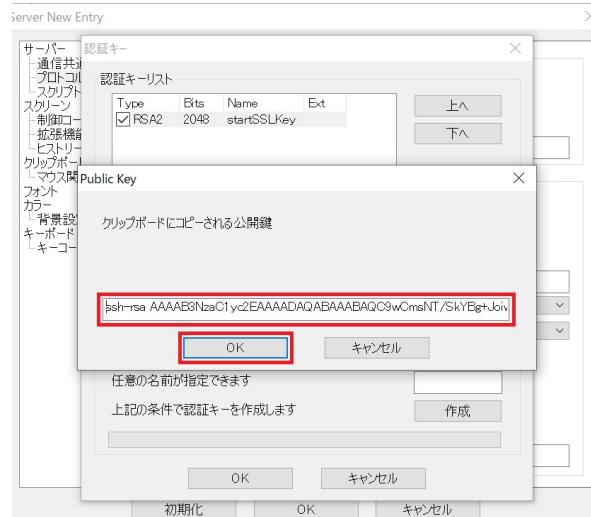
▲図 3.5 [OK] をクリック

無事に SSH のキーペア（秘密鍵と公開鍵）が作成されて、[認証キーリスト] に [startSSLKey] が表示されました。[公開鍵] をクリックしてください。（図 3.6）



▲図 3.6 [startSSLKey] が表示されたら [公開鍵] をクリック

公開鍵が表示されました。この後すぐに使いますので、[クリップボードにコピーされる公開鍵] で表示された公開鍵 (ssh-rsa から始まる文字列) をまるごとコピーして、メモ帳などにペーストしておきましょう。公開鍵をメモしたら [OK] をクリックして閉じます。



▲図 3.7 表示された公開鍵（文字列）はまるごとコピーしてメモ帳にペーストしておこう

あとは [キャンセル] を繰り返しクリックして、起動中の RLogin は一旦閉じてしまって構いません。RLogin は、後でサーバへ入るときに使いますので、デスクトップの「rlogin_x64」フォルダと、その中にある「RLogin.exe」をごみ箱へ捨てないように注意してください。メモ帳にメモした公開鍵も、消してしまわないようご注意ください。

3.1.3 Mac でターミナルを準備する

Mac^{*4}を使っている方は、最初から「ターミナル」(図 3.8) というソフトがインストールされていますのでそちらを利用しましょう。

^{*4} 2020 年 2 月以前にリリースされた macOS を対象とします



▲図 3.8 最初からインストールされている「ターミナル」を使おう

ターミナルがどこにあるのか分からぬときは、Mac の画面で右上にある虫眼鏡のマークをクリックして、Spotlight で「ターミナル」と検索（図 3.9）すれば起動できます。



▲図 3.9 どこにあるのか分からなかつたら Spotlight で「ターミナル」と検索

3.1.4 Mac で SSH のキーペア（秘密鍵・公開鍵）を作成する

ターミナルを起動したら、キーペアを作成します。Mac を使っている方は、ターミナルで次のコマンドを実行してください。⁵先頭の「\$」を入力する必要はありません。

```
$ ssh-keygen -N '' -f ~/Desktop/startSSLKey
```

次のように表示されたらキーペア（秘密鍵・公開鍵）の作成は完了です。

```
$ ssh-keygen -N '' -f ~/Desktop/startSSLKey
Generating public/private rsa key pair.
Your identification has been saved in /home/mochikoAsTech/Desktop/startSSLKey.
Your public key has been saved in /home/mochikoAsTech/Desktop/startSSLKey.pub.
The key fingerprint is:
a2:52:43:dd:70:5d:a8:4f:77:47:ca:f9:69:79:14:48 mochikoAsTech@mochikoMacBook-Air.local.
The key's randomart image is:
+--[ RSA 2048]----+
| . . ooE. |
| . + o . . |
| . . . . +. |
| . . . = o |
| o . So . . +o |
| . o . . +o |
| . . . . |
| . . . |
| . . . |
+-----+
```

作成した秘密鍵は、オーナー以外が使えないよう chmod というコマンドで読み書き権限を厳しくしておきます。

```
$ chmod 600 ~/Desktop/startSSLKey
```

ホームディレクトリに秘密鍵（startSSLKey）と、公開鍵（startSSLKey.pub）ができるあがっているはずです。cat（キャット）コマンド⁶で公開鍵を表示してみましょう。

⁵ ssh-keygen コマンドは名前のとおり、SSH の鍵（key）を生成（generate）するコマンドです。-f オプションでは、生成する鍵のファイル名を指定しています。～（チルダ）はホームディレクトリを表しますので、-f ~/Desktop/startSSLKey は「/Users/<ユーザ名>/Desktop」のフォルダの中に「startSSLKey」という名前の鍵を作って、という意味です。-N ''は空のパスフレーズを指定しています

⁶ cat は猫ではなく「conCATenate files and print on the standard output」の略です

```
$ cat ~/startSSLKey.pub  
ssh-rsa AAAAB3NzaC1yc2E=Unidb+6FjiLw== mochikoAsTech@mochikoMacBook-Air.local
```

この後すぐに使いますので、表示された公開鍵（ssh-rsa から始まる文字列）をまるごとコピーして、メモ帳などにペーストしておきましょう。

以上で事前準備は完了です。お待たせしました。いよいよサーバを立てましょう。

【コラム】SSH の秘密鍵にパスフレーズは設定すべき？

秘密鍵に「[パスフレーズ]」を設定しておくと、鍵を使って SSH でサーバに入ろうとしたとき、「鍵を発動するにはパスフレーズを叫べ…！」という感じでパスフレーズを聞かれます。

つまり、もしあなたの秘密鍵が盗まれて誰かに勝手に使われそうになんて、パスフレーズを設定していれば鍵の悪用が防げます。スマホ本体が盗まれてしまって、暗証番号が分からなければロック画面が解除できず、勝手に使えないのと同じです。

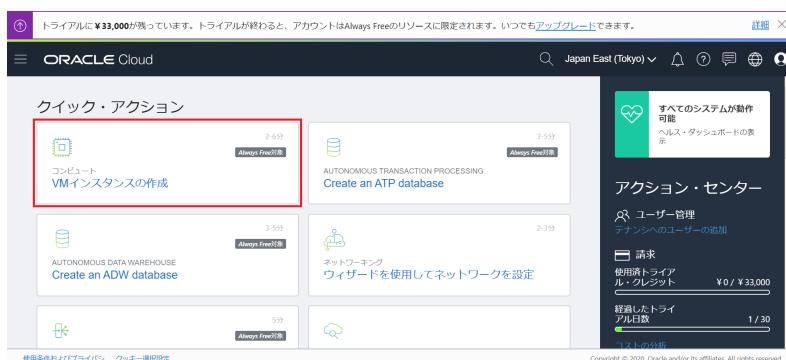
これは「あなたは何を持っているのか」「あなたは何を知っているのか」「あなたは誰なのか」という複数の要素の中から、2つを用いることで認証の強度を高める「二要素認証」と呼ばれる考え方です。「あなたは秘密鍵を持っている」「あなたはパスフレーズを知っている」という2つの要素を組み合わせることで、単要素での認証よりも強度が高まります。ちなみに「あなたは誰なのか」は指紋認証や顔認証ですね。

ですが「パスワード認証じゃなくて鍵認証なのに、やっぱりパスフレーズが必要なの…？」という具合に、初心者を混乱に陥れやすいので、本書では秘密鍵をパスフレーズなしで作って使用します。

パスフレーズは「設定していれば絶対に安心！」というものではありませんが、上記の理由から、本来であれば設定した方がいいものです。後で「やっぱり設定しておこう」と思ったら、一度作成した秘密鍵に対して、後からパスフレーズを設定することも可能です。なお手元の秘密鍵に、後からパスフレーズを設定しても、サーバ側での変更は特に必要ありません。

3.2 コンピュートでサーバを立てる

それでは下準備ができたので、Oracle Cloud のコンソールに戻ってサーバを立てましょう。コンソールにサインインしたら、[VM インスタンスの作成] をクリック（図 3.10）します。ちなみに Oracle Cloud にはデータベース管理やストレージなど、さまざまなサービスがありますが、クラウドサーバや物理サーバなどの**サーバが立てられるサービスは「コンピュート」と呼ばれています**。そして Oracle Cloud では**サーバのことをインスタンス**と呼びます。ここから先でインスタンスと書いてあつたら「サーバのことだな」と思ってください。



▲図 3.10 [VM インスタンスの作成] をクリック

[コンピュート・インスタンスの作成] が表示されたら、[インスタンスの命名] に [startSSLInstance] と入力します。（図 3.11）その下の [オペレーティング・システムまたはイメージ・ソースを選択します] は、何も変更せず [Oracle Linux 7.7] のままで構いません。



▲図 3.11 [インスタンスの命名] に [startSSLInstance] と入力

ところでこの「オペレーティング・システム」とはいったい何のことでしょうか？

3.2.1 OS は Oracle Linux 7.7 を使おう

パソコンには OS（オペレーティング・システム）という基本ソフトが入っていて、Word や Excel、Chrome といったソフトはその OS の上で動いています。皆さんのパソコンにも「Windows 10」や「Mac OS X Lion」などの OS が入っていますよね。

そしてパソコンと同じようにサーバにも「Linux」や「Windows Server」といったサーバ用の OS があります。サーバを立てるときには Linux を選択することが多いのですが、この Linux の中にもさらに「RHEL (Red Hat Enterprise Linux)」や「CentOS」、「Ubuntu」などいろいろなディストリビューション（種類）があります。

本書では、OS はデフォルトの「Oracle Linux 7.7」を使用します。Oracle Linux なら Oracle Cloud のツールがあらかじめ入っていますので、Oracle Cloud でインスタンスを立てるときは OS は Oracle Linux にすることをお勧めします。Oracle Linux は Red Hat 系のディストリビューションですので、RHEL や CentOS のサーバを使ったことがある方なら違和感なく使えると思います。

2020 年 1 月時点で、Oracle Linux には次の 2 種類があります。

- Oracle Linux 6.10
- Oracle Linux 7.7

Oracle Linux 6.10 は CentOS 6 と同じ RHEL6 系、Oracle Linux 7.7 は CentOS 7 と同じ RHEL7 系なので、使い勝手もほぼ同じです。

3.2.2 作っておいた SSH の公開鍵を設置しよう

さらに下にスクロールします。[SSH キーの追加] は、[SSH キーの貼付け] を選択します。先ほどメモ帳にメモしておいた公開鍵を [SSH キー] にペーストします。公開鍵は改行を含まず、先頭の「ssh-rsa」から末尾の「<ユーザ名>@<ホスト名>」のようなコメントまで、まるごと 1 行です。(図 3.12)



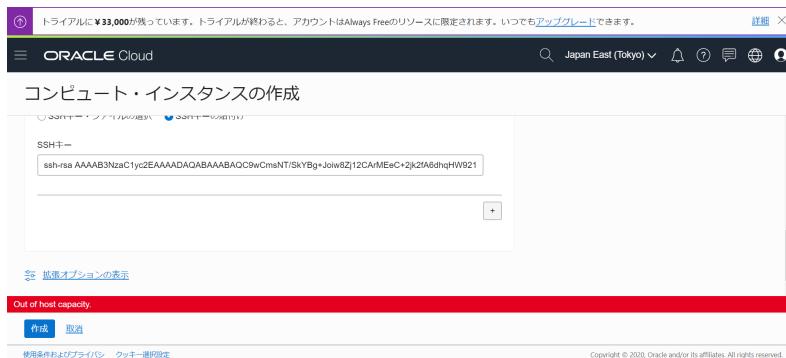
▲図 3.12 [SSH キーの貼付け] を選択してメモしておいた公開鍵をペースト

公開鍵をペーストしたら [作成] をクリックします。

3.2.3 〈トラブル〉 "Out of host capacity."が起きたらどうすればいい？

さて、元気よく [作成] をクリックしたのに、真っ赤な [Out of host capacity.] が表示されてしまった…という方が一定数いらっしゃると思います。大丈夫、あなたは悪くありません。いま理由を説明するので落ち着いてください。「そんなの表示されなかったよ？」という方は、この〈トラブル〉は読み飛ばして、この先の「3.2.5 サーバが起動するまで待とう」までジャンプしてください。

3.2 コンピュートでサーバを立てる



▲図 3.13 "Out of host capacity."と表示されて何も起きない！

"Out of host capacity."は、直訳すると「ホスト容量が不足しています」という意味ですが、ホストってなんでしょう？

あなたがいま Oracle Cloud で立てようとしたインスタンスは、家でいうと「一軒家」ではなく、マンションの 101 号室や 403 号室のような「各部屋」にあたります。このときマンションの建物をホストサーバ、各部屋をゲストサーバと呼びます。



▲図 3.14 マンションの建物をホストサーバ、各部屋をゲストサーバと呼ぶ

「ホストの容量が不足している」ということは…つまり、あなたが Oracle Cloud の無料マンションに入居しようとしたら、「ごめんね、無料マンションは大人気でいま空き部屋がないの」と断られてしまった、という状況なのです。

Oracle Cloud の Always Free は有効期限なしでずっと無料で使える、とても魅力的なサービスです。そのため Oracle Cloud 側も定期的に新築マンションを追加しているもの

の、頻繁にリソース不足に陥ってはこういう状況になるようです。

この"Out of host capacity."が発生してしまった場合、次のどちらかが起きてホストの容量不足が解消しない限り、Always Free の枠でインスタンスは立てられません。

- 自分以外のユーザがインスタンスを解約してリソースを開放する
- Oracle Cloud がリソースを増やす

ですが、Always Free とは別に、我々には 30 日間だけ有効な\$300 の無償クレジットが与えられています。たとえ無料マンションが満室でも、有料マンションなら空きがあります。30 日経ったら消えてしまう\$300 のお小遣いを握りしめて、次の方法で有料マンションのお部屋を借りにいきましょう！

3.2.4 Always Free ではなく無償クレジットの枠でサーバを立てよう

次の手順は、"Out of host capacity."が表示された人だけ実施してください。

"Out of host capacity."を回避するため、少し上にスクロールして [シェイプ、ネットワークおよびストレージ・オプションの表示] をクリック（図 3.15）しましょう。もともと選択していたのは [Always Free 対象] のマークが付いた [VM.Standard.E2.1.Micro (仮想マシン)] という種類のインスタンスでした。



▲図 3.15 [シェイプ、ネットワークおよびストレージ・オプションの表示] をクリック

Oracle Cloud では、マシンスペックごとに「シェイプ」という区分^{*7}があります。インスタンスのシェイプを、いま選択されている [VM.Standard.E2.1.Micro] から変更するため、[シェイプの変更] をクリックしてください。

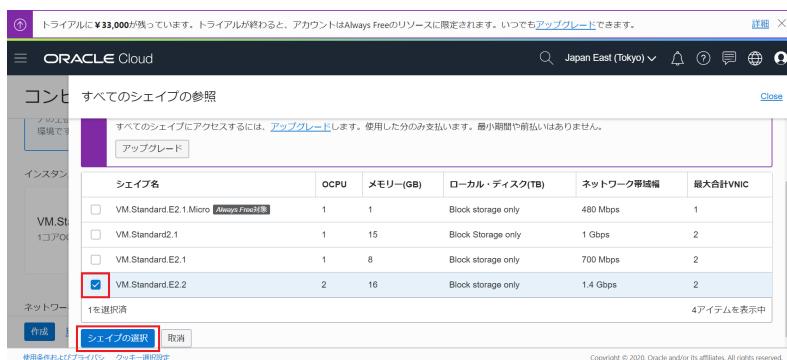
*7 AWS のインスタンスタイプと同じものだと思ってください

3.2 コンピュートでサーバを立てる



▲図 3.16 「シェイプの変更」をクリック

OCPU^{*8}が 2、メモリが 16GB の [VM.Standard.E2.2]^{*9}にチェックを入れて、「シェイプの選択」をクリックしましょう。

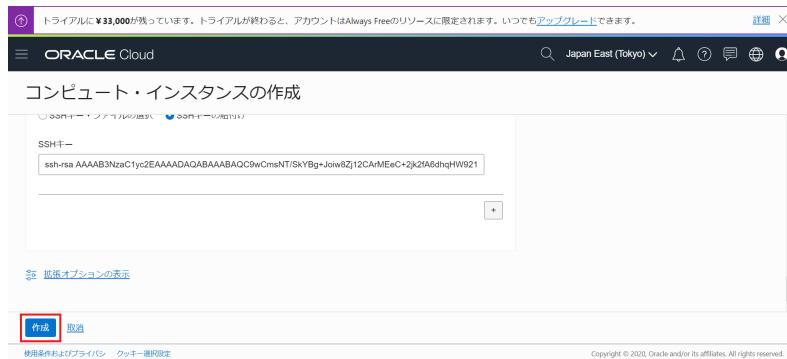


▲図 3.17 「VM.Standard.E2.2」に変更して「シェイプの選択」をクリック

それ以外は何も変更せずに、いちばん下の「作成」をクリックします。

^{*8} OCPU は Oracle Compute Units の略で、ごく簡単に言うと CPU コア数の単位です。OCPU1 つは、vCPU（仮想 CPU）2 つに相当しますので、もし「AWS の EC2 で vCPU が 4 のインスタンスを使っている。同等スペックのインスタンスを用意してほしい」と頼まれたら、Oracle Cloud では OCPU が 2 のシェイプを選べば大丈夫です。ただし CPU の単純比較だけではなく、他にもメモリやストレージなど、必要なスペックを考えて適したシェイプを選ぶようにしましょう

^{*9} シェイプの名前は、まず接頭辞が「VM」なら仮想サーバ（Virtual Machine）、「BM」なら物理サーバ（Bare Metal）を表しています。その後ろの単語は「Standard」（汎用）や「DenseIO」（高密度 I/O）といった特徴、3 番目の「E2」や「2」はシェイプの世代、最後の「2」や「8」は OCPU の数を表しています



▲図 3.18 [作成] をクリック

3.2.5 サーバが起動するまで待とう

オレンジ色で「[プロビジョニング中...]」と表示（図 3.19）されたら、サーバが起動するまでそのまま数分待ちましょう。



▲図 3.19 [プロビジョニング中...] と表示されたら数分待つ

サーバが起動すると、表示が緑色の「実行中」に変わります（図 3.20）。おめでとうございます。サーバが立てられました！



▲図 3.20 「実行中」に変わった！ サーバが立った！

【コラム】Oracle Cloud のコンピュートの金額計算方法

ところで、「VM.Standard.E2.2」というシェイプのインスタンスをまるまる1ヶ月使ったら、一体いくら分になるのでしょうか？ うっかり\$300を超てしまわないか、ちょっと心配なので計算してみましょう。

コンピュートの価格表^{*10}を見てみると、[Compute - Standard - E2] は [\$0.03]^{*11}と書いてあります。これは [Pay as You Go (OCPU Per Hour)] と書いてあるとおり、1OCPU ごとに 1 時間あたりかかる金額です。^{*12}

「VM.Standard.E2.2」は OCPU が 2 なので、\$0.03*2 で 1 時間あたり \$0.06 かかることが分かります。1 ヶ月を 744 時間 (24 時間*31 日) として、\$0.06*744 時間で \$44.64 です。つまり「VM.Standard.E2.2」を 1 台立てたくらいでは、\$300 の無償クレジットを使い切ることはないので安心しましょう。ちなみに Oracle Cloud では \$1 は 120 円換算^{*13}なので、\$44.64 は日本円だと 5356.8 円ですね。

^{*12} <https://www.oracle.com/jp/cloud/compute/pricing.html>

^{*13} 2020 年 2 月時点の金額

^{*14} ちなみに AWS は、同スペックのインスタンスでもリージョンごとに価格が異なりますが、Oracle Cloud はどのリージョンでも同一の価格です

^{*15} 2020 年 2 月時点のレート

【コラム】 Oracle Cloud と AWS はどっちが安い？

Oracle Cloud は他のクラウドに比べて価格が安いのが特徴のひとつです。どれくらい安いのか、Oracle Cloud と AWS でサーバの価格を比較してみましょう。

例えば同スペックの VM.Standard.E2.1 (Oracle Cloud) と m5a.large (AWS) を比較すると、Oracle Cloud の価格は AWS の 3 分の 1 以下になります。(表 3.1)

▼表 3.1 Oracle Cloud と AWS の価格比較

	Oracle Cloud	AWS
インスタンスの種類	VM.Standard.E2.1	m5a.large
CPU	OCPU:1 (vCPU:2 相当)	vCPU:2
メモリ	8GB	8GB
1 時間あたり	\$0.03	\$0.112
月額	2678.4 円	9999.36 円

シェアトップを独走する AWS に対して、後発のクラウドは勝つためにコスト面や性能面でそれぞれ大きなメリットを打ち出してきています。AWS が最適なであれば AWS を選択すべきですが、「みんなが使っているから」というだけの理由で、あまり深く考えずに AWS を使っているのであれば、他のクラウドにも目を向けてみることをお勧めします。

3.2.6 接続先サーバの IP アドレス

無事にサーバが立ち上がったら、IP アドレスを確認しておきましょう。

先ほど作成したインスタンスの [startSSLInstance] で、[プライマリ VNIC 情報] (図 3.21) にある [パブリック IP アドレス] をメモ (表 3.2) してください。

3.3 サーバに SSH でログインしよう



▲図 3.21 [プライマリ VNIC 情報] の [パブリック IP アドレス] をメモしておこう

▼表 3.2 インスタンスの [パブリック IP アドレス]

例	パブリック IP アドレス
	140.238.33.51

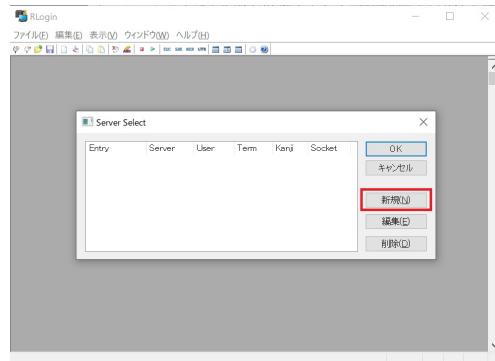
それではメモした IP アドレスを使ってサーバにログインしてみましょう。

3.3 サーバに SSH でログインしよう

立てたばかりのサーバに、ターミナルを使って SSH でログインします。Windows、Mac の順番で手順を説明します。

3.3.1 Windows の RLogin を使ってサーバにログインしよう

Windows のパソコンを使っている方は、RLogin を起動（図 3.22）してください。起動したら [新規] をクリックします。



▲図 3.22 RLogin が起動したら [新規] をクリック

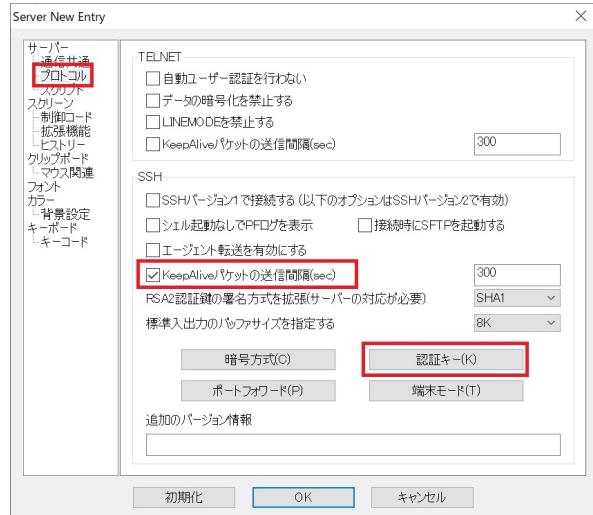
まずは [エントリー (上)] に [startSSLInstance] と入力します。[コメント (下)] は入力しなくて構いません。続いて [ホスト名 (サーバー IP アドレス)] に、先ほどメモした [パブリック IP アドレス] を入力（図 3.23）します。[ログインユーザー名] には [opc] と入力してください。opc というのは Oracle Linux のインスタンスを作成すると、最初から存在しているデフォルトユーザです。



▲図 3.23 [ホスト名 (サーバー IP アドレス)] と [ログインユーザー名] を入力

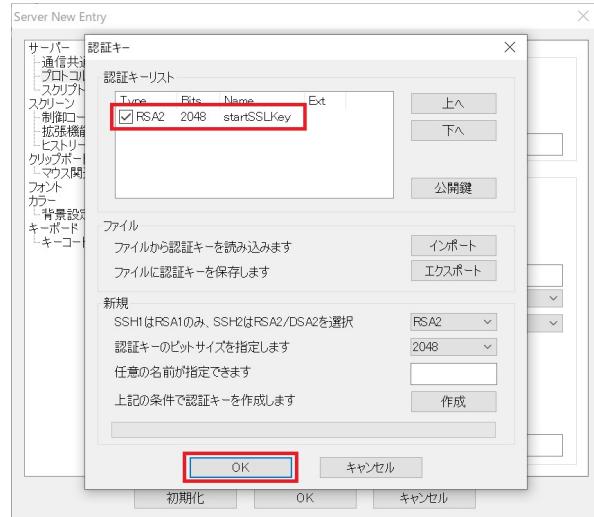
次に左メニューで [プロトコル] を選択（図 3.24）したら、[KeepAlive パケットの送信間隔 (sec)] にチェックを入れておきます。チェックを入れておくと、ターミナルをしばらく放っておいても接続が勝手に切れません。続いて [認証キー] をクリックします。

3.3 サーバに SSH でログインしよう



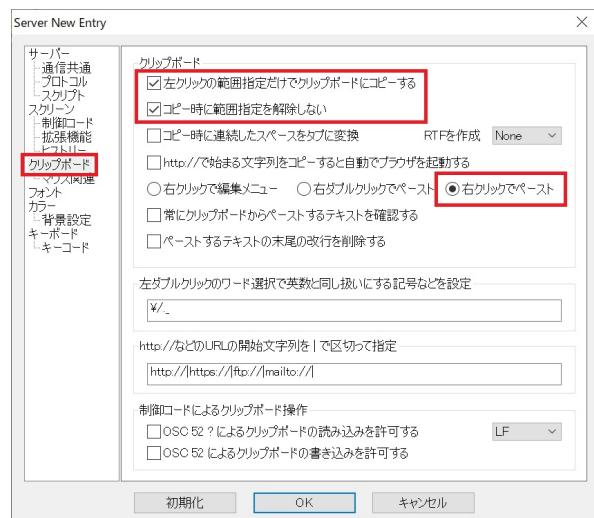
▲図 3.24 チェックを入れて「認証キー」をクリック

【認証キー】リストで [startSSLKey] にチェックが入っていることを確認（図 3.25）します。このリストはログイン時に使用する鍵のリストです。キーケースに自宅と実家の鍵をぶらさげておくように、複数の鍵をリストに入れておくことも可能です。チェックが入っていたら [OK] をクリックして閉じましょう。



▲図 3.25 [startSSLKey] にチェックが入っていることを確認

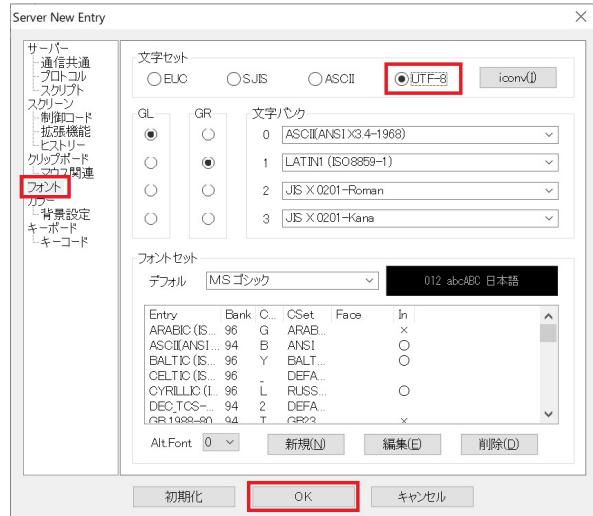
続いて左メニューで [クリップボード] を選択（図 3.26）したら、[左クリックの範囲指定だけでクリップボードにコピーする] と [コピー時に範囲指定を解除しない] にチェックを入れて [右クリックでペースト] を選択します。



▲図 3.26 右クリックや左クリックの設定

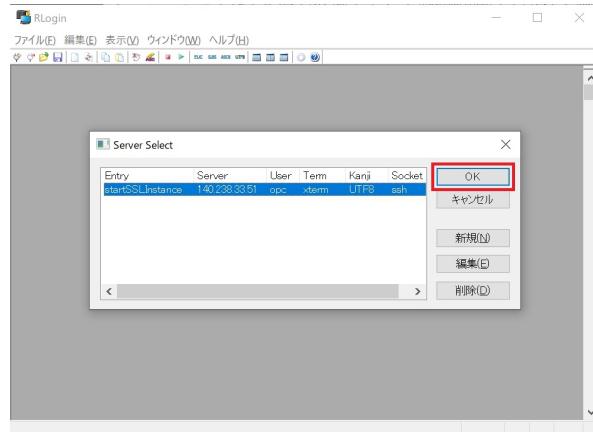
3.3 サーバに SSH でログインしよう

次に左メニューで [フォント] を選択（図 3.27）したら、文字セットを [UTF-8] に変更します。すべて設定できたら [OK] をクリックしてください。



▲図 3.27 文字セットを [UTF-8] に変更

これで設定が保存できました。今やった設定は最初の 1 回のみで、2 回目以降は不要です。それでは [OK] をクリック（図 3.28）して、サーバにログインしましょう。



▲図 3.28 設定が保存できたので [OK] をクリックしてサーバにログイン

すると初回のみ、この【公開鍵の確認】が表示（図3.29）されますので、【接続する】をクリック^{*14}します。下部の【この公開鍵を信頼するリストに保存する】にチェックが入っていると、RLogin がサーバ固有のフィンガープリント^{*15}を覚えてくれます。そして次回以降は「これはこのサーバで過去に信頼した公開鍵と同じだ」と判断して、RLogin がそのままサーバへ接続させてくれます。



▲図3.29 【公開鍵の確認】が表示されたら【接続する】をクリック

続いて【信頼するホスト鍵のリストを更新しますか？】と聞かれたら【はい】をクリック（図3.30）してください。

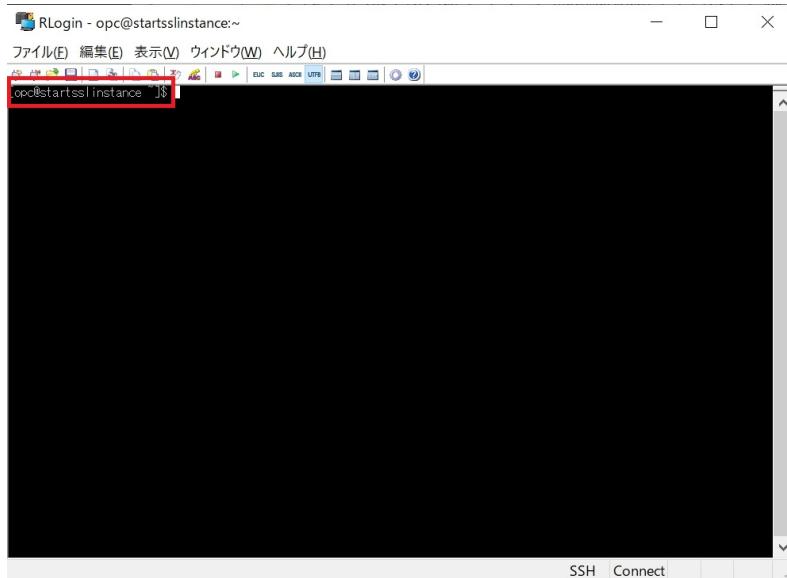


▲図3.30 【信頼するホスト鍵のリストを更新しますか？】と表示されたら【はい】をクリック

^{*14} 今後、別のパソコンから同じサーバへ接続する際には、元のパソコンに保存されているサーバのフィンガープリントと、初回接続時に表示されるフィンガープリントを比較することで、DNS キャッシュポイズニングなどの攻撃によって、他のサーバに誘導されていないか確認できます

^{*15} 公開鍵のハッシュ値のことをフィンガープリント（指紋）と呼びます

[opc@startsslinstance] と表示（図 3.31）されたら無事サーバにログインできています。自分でサーバを立てて、SSH でのログインすることができましたね。おめでとうございます！



▲図 3.31 [opc@startsslinstance] と表示されたら成功！

3.3.2 <トラブル> サーバにログインできない！

もし [opc@startsslinstance] と表示されず、代わりに [SSH2 User Auth Failure "publickey,gssapi-keyex,gssapi-with-mic" Status=1004 Send Disconnect Message... gssapi-with-mic] というようなエラーメッセージが表示（図 3.32）されてしまったら、これは「鍵がない人は入れないよ！」とお断りされている状態です。[認証キー] リストで [startSSLKey] にチェックが入っていないか、あるいは誤って別の秘密鍵を選択しているものと思われますので、[認証キー] の設定を確認してみてください。



▲図 3.32 このエラーが表示されたら [認証キー] を確認しよう

[接続済みの呼び出し先が一定の時間を過ぎても正しく応答しなかったため、接続できませんでした。] というエラーメッセージが表示（図 3.33）されてしまった場合は、[ホスト名（サーバー IP アドレス）] に書いた [パブリック IP アドレス] が間違っているものと思われます。[ホスト名（サーバー IP アドレス）] の IP アドレスを確認してみてください。



▲図 3.33 このエラーが表示されたら [ホスト名（サーバー IP アドレス）] の IP アドレスを確認しよう

3.3.3 Mac のターミナルを使ってサーバにログインしよう

Mac を使っている方は、ターミナル（図 3.34）を起動してください。



▲図 3.34 「ターミナル」を起動しよう

起動したターミナルで次の文字を入力したら Return キーを押します。「パブリック IP アドレス」の部分は先ほどメモした「パブリック IP アドレス」に書き換えてください。
-i オプションは「サーバにはこの鍵を使って入ります」という意味です。「startSSLKey」を保存した場所がデスクトップ以外だった場合はこちらも適宜書き換えてください。

```
$ ssh opc@パブリック IP アドレス -i ~/Desktop/startSSLKey
```

初回のみ次のようなメッセージが表示されますが、「yes」と打って Return キー^{*16}を押してください。

```
Are you sure you want to continue connecting (yes/no)?
```

【opc@startsslinstance】と表示されたら、SSH でサーバにログインできています。おめでとうございます！今後は、いまやったのと同じやり方をそのまま繰り返せば、サーバにログインできます。

^{*16} yes と打った理由は、P62 の注釈で説明しています

3.4 NGINXをインストールしよう

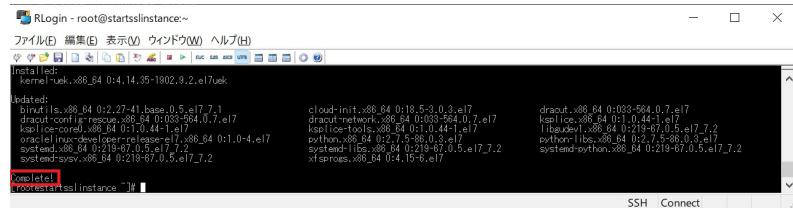
それではサーバにログインできたので、必要なミドルウェアをインストールしていきましょう。最初にrootになっておいてください*17。

```
$ sudo su -
```

インストールするときはyum（ヤム）というコマンドを使います。先ずはyumで色々アップデートしておきましょう。Windows アップデートみたいなものです。

```
# yum update -y
```

画面にたくさん文字が流れ、少し時間がかかりますが、最後に「[Complete!]」と表示（図3.35）されたら問題なく完了しています。ちなみに-yオプションはYESを意味するオプションです。-yオプションをつけないで実行すると「これとこれをアップデートするけどいい？ダウンロードサイズとインストールサイズはこれくらいだよ」という確認が表示され、yと入力してEnterキーを押さないとアップデートされません。



▲図3.35 最後に「[Complete!]」と表示されたらアップデートは完了

続いてウェブサーバのNGINX*18をインストールします。2020年1月現在の安定バ-

*17 以降は、例として書いてある部分のプロンプトが、「\$」だったらopcのような一般ユーザでコマンドを実行、「#」だったらrootでコマンドを実行してください。例に「\$」や「#」が書いてあっても、ターミナルで「\$」や「#」を自分で入力する必要はありません

*18 NGINXと書いて「えんじんえっくす」と読みます。ウェブサーバのミドルウェアの中でNGINXは順調にシェアを伸ばし、2019年4月にとうとうApacheを抜いてシェア1位になりました。<https://news.mynavi.jp/article/20190424-813722/>

ジョン^{*19}である 1.16 系をインストールしたいので、yum のリポジトリ（どこから NGINX をダウンロードしてくるか）に NGINX 公式を追加しましょう。

```
# rpm -ivh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7.ngx.noarch.rpm ← 実際は改行なしで 1 行
# cat /etc/yum.repos.d/nginx.repo

↓ 以下が表示されれば OK
# nginx.repo

[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/7/$basearch/
gpgcheck=0
enabled=1
```

yum で NGINX をインストールします。

```
# yum install -y nginx
```

[Complete!] と表示されたらインストール完了です。バージョン情報を表示することで、ちゃんとインストールされたか確認してみましょう。

```
# nginx -v
nginx version: nginx/1.16.1 ← バージョン情報が表示されればインストールできている
```

ちょっと分かりにくいかも知れませんが、パソコンに Microsoft Excel をインストールしたら「表計算というサービスが提供できるパソコン」になるのと同じで、サーバにこの NGINX をインストールすると「リクエストに対してウェブページを返すサービスが提供できるサーバ」、つまりウェブサーバになります。今回は NGINXを入れましたが、ウェブサーバのミドルウェアは、Apache をはじめとして他にも色々な種類があります。

インストールが終わったので、サーバを再起動した場合も NGINX が自動で立ち上がりてくるよう、自動起動の設定もオンにしておきましょう。systemctl コマンド^{*20}で、NGINX の自動起動を有効 (enable) にします。

^{*19} サーバに入っている NGINX のバージョンがいくつなのか？ という情報は大切です。今後、あなたが NGINX の設定ファイルを書こうと思って調べたとき、例えば 1.12 系の設定方法を参考にしてしまうと、1.16 系の NGINX の環境では上手く動かない可能性があります。

^{*20} 自動起動の設定は、CentOS 6 系だと chkconfig コマンドで行なっていましたが、7 系では systemctl コマンドになっています

```
# systemctl enable nginx
# systemctl is-enabled nginx
enabled ←有効になったことを確認
```

3.5 Firewalld で HTTP と HTTPS を許可しよう

続いてサーバの中で動いているファイアウォールの設定を変更します。まずは現状、何がファイアウォールを通れるようになっているのか確認^{*21}してみましょう。

```
# firewall-cmd --list-services
dhcpv6-client ssh
```

dhcpv6-client と ssh は通っていいけれど、それ以外は誰であろうと通さないぞ！ という設定になっています。このままではブラウザでサイトを見ようとしても、ウェブページを返してくれるはずの NGINX まで、HTTP や HTTPS のリクエストが届きません。そこで次のように、許可対象に http と https を追加します。追加したら、変更が反映されるようリロードしてください。それぞれ [success] と表示されたら成功しています。

```
# firewall-cmd --add-service=http --permanent
success

# firewall-cmd --add-service=https --permanent
success

# firewall-cmd --reload
success
```

これでファイアウォールの設定が変更できたので、もう一度、ファイアウォールの設定を確認してみましょう。ファイアウォールを通っていい対象に、http と https が追加されていれば問題ありません。

```
# firewall-cmd --list-services
dhcpv6-client http https ssh
```

これで、HTTP でサイトを見るためのウェブサーバの設定はおしまいです。

^{*21} Oracle Linux 7、及び Oracle Cloud の CentOS 7 では firewalld がデフォルトで有効化されています。
https://docs.oracle.com/cd/E77565_01/E54670/html/o17-firewalld-cfg.html

3.6 なぜか HTTP でサイトが見られない

ウェブサーバも立てたし、サーバの中のファイアウォールに穴も空けました。これで準備完了！ サーバを立てたときにメモした「パブリック IP アドレス」をコピーしたら、ブラウザのアドレスバーにペーストして、サイトを開いてみましょう。

するとしばらくぐるぐるした後で、【接続がタイムアウトしました】と表示されてしまいました。（図 3.36）



▲図 3.36 なぜか HTTP でサイトが表示されない…

403 や 404、あるいは 502 などのステータスコードも返ってきていないので、そもそも NGINX までたどり着けていないようです。

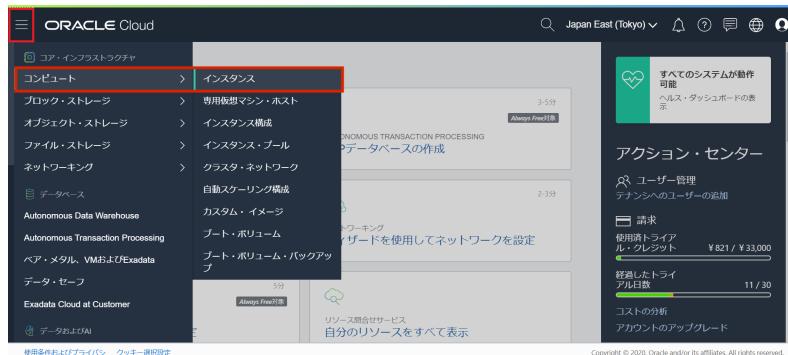
3.6.1 サーバの手前にあるファイアウォールにも穴を空けよう

先ほどサーバの中にあるファイアウォールの設定を変更して、HTTP と HTTPS が通れるようにしましたが、実はサーバの中だけでなく、サーバに辿り着く手前にも、もう 1 つファイアウォールがいます。サイトが表示されなかったのは、「ウェブページを見せて！」というリクエストが、**サーバの手前のファイアウォールで阻まれていたため**なのです。サーバの手前にあるファイアウォールにも穴を空けて、HTTP と HTTPS がサーバまでたどり着けるようにしましょう。

再び Oracle Cloud のコンソールに戻って、左上の【ハンバーガーメニュー】から【コ

第3章 サーバを立てて HTTP でサイトを見てみよう

ンピュート] の [インスタンス] を開きます。(図 3.37)



▲図 3.37 [ハンバーガーメニュー] から [コンピュート] の [インスタンス] を開く

インスタンスの一覧が表示されるので [startSSLInstance] をクリックします。(図 3.38)



▲図 3.38 [startSSLInstance] をクリック

[パブリック・サブネット] をクリックします。(図 3.39)

3.6 なぜか HTTP でサイトが見られない

実行中

インスタンス情報

可用性ドメイン: iomt/AP-TOKYO-1-AD-1
フルト・ドメイン: FAULT-DOMAIN-3
リージョン: ap-tokyo-1
シエイフ: VM.Standard.E2.2
仮想クラウド・ネットワーク: VirtualCloudNetwork-20200120-2319
メンテナンス再起動: -

プライマリVNIC情報

プライベートIPアドレス: 10.0.0.2
パブリックIPアドレス: 140.238.33.51
ネットワーク・セキュリティ・グループ: None [編集]

内部FQDN: startdnsinstance... [表示] [コピー]
サブネット: [パブリック・サブネット]

このインスタンスのトラフィックは、関連付けられた[サブネット](#)のセキュリティ・リストおよびVNICのネットワーク・セキュリティ・グループ

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

▲図 3.39 [パブリック・サブネット] をクリック

もう一度、[パブリック・サブネット] をクリックします。(図 3.40)

使用可能

CIDRブロック: 10.0.0.0/16
コンバートメント: startdns01 (ルート)
作成日: 2020年1月20日(月) 14:55:54 UTC

OCID: ..._tpoa [表示] [コピー]
デフォルト・ルート表: Default Route Table for VirtualCloudNetwork-20200120-2319
トポ:

DNSドメイン名: vcn.oracledev.com

startdns01 (ルート) コンパートメント内のサブネット

サブネットの作成				
名前	状態	CIDRブロック	サブネット・アクセス	作成日
[パブリック・サブネット]	● 使用可能	10.0.0.0/24	パブリック (リージョナル)	2020年1月20日(月) 14:55:56 UTC

1アイテムを表示中 < ページ1 >

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

▲図 3.40 もう一度、[パブリック・サブネット] をクリック

[セキュリティ・リスト] の中にある [Default Security List for VirtualCloudNetwork ~] をクリック (図 3.41) します。この [セキュリティ・リスト] が、サーバの手前にいるファイアウォールです。

第3章 サーバを立てて HTTP でサイトを見てみよう

The screenshot shows the Oracle Cloud Infrastructure interface. At the top, it displays network details: CIDR ブロック: 10.0.0.0/4, 仮想ルーター MAC アドレス: 00:00:17:42:14:B4, and サブネット・タイプ: リージョナル. On the right, it shows DNS ドメイン名: subnet... 直近 コレ, サブネット・アクセス: パブリック / サブネット, DHCP オプション: Default DHCP Options for VirtualCloudNetwork-20200120-2319, and ルート表: Default Route Table for VirtualCloudNetwork-20200120-2319. Below this, there's a table titled 'セキュリティ・リスト' (Security List) with one item: 'Default Security List for VirtualCloudNetwork-20200120-2319'. The status is '使用可能' (Available). The table has columns: 名前 (Name), 状態 (Status), コンバートメント (Conversion), and 作成日 (Created Date). The item is listed as 'Default Security List for VirtualCloudNetwork-20200120-2319' with '使用可能' (Available) status, 'startdns01 (ルート)' conversion, and '2020年1月20日(月) 14:55:54 UTC' creation date.

▲図 3.41 [Default Security List for VirtualCloudNetwork～] をクリック

[イングレス・ルール] で、[イングレス・ルールの追加] をクリック（図 3.42）します。

The screenshot shows the Oracle Cloud Infrastructure interface for creating an ingress rule. The title is 'Ingress Rule'. There are two tabs: 'Ingress Rule (3)' (selected) and 'Egress Rule (1)'. A button 'Create New Rule' is highlighted with a red box. Below it is a table with columns: 'Source IP Address' (ソース IP アドレス), 'Protocol' (プロトコル), 'Source Port Range' (ソースポート範囲), 'Destination Port Range' (宛先ポート範囲), 'Type and Code' (タイプとコード), and 'Allow' (許可). Two rules are listed: one for TCP port 22 (SSH) and another for ICMP port 3-4.

▲図 3.42 [イングレス・ルールの追加] をクリック

[ソース CIDR] に [0.0.0.0/0]^{*22}を入力します。[宛先ポート範囲] には [80,443]^{*23}を入力します。（図 3.43）どちらも入力できたら、[イングレス・ルールの追加] をクリックします。

*²² ソース CIDR は接続元の IP アドレス範囲のこと、0.0.0.0/0 はすべての IP アドレスを指します。つまり接続元がどんな IP アドレスでもファイアウォールを通します、ということですね

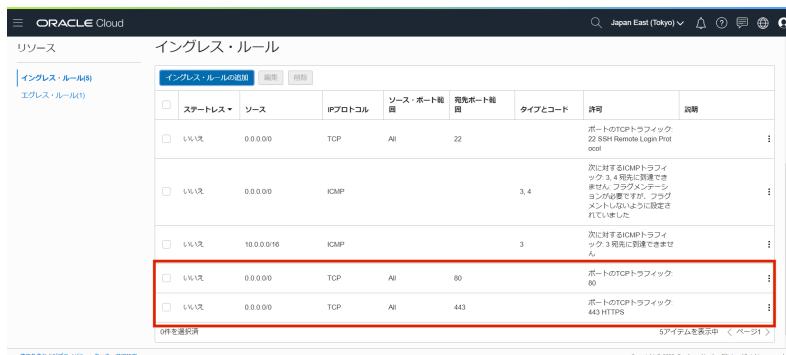
*²³ ポート番号とは、サーバというやつや、その手前のファイアウォールという壁についているドアの番号のようなものだと思ってください。同じサーバを訪問するときでも SSH は 22 番のドアを、HTTP は 80 番のドアを、HTTPS は 443 番のドアを通ります

3.6 なぜか HTTP でサイトが見られない



▲図 3.43 [ソース CIDR] に [0.0.0.0/0]、[宛先ポート範囲] には [80,443] を入力

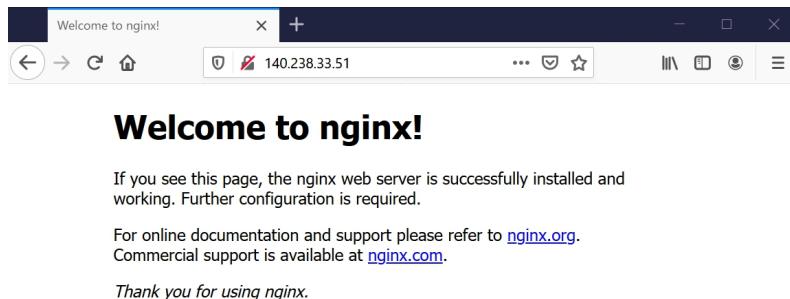
[イングレス・ルール] に、HTTP (80 番ポート) と HTTPS (443 番ポート) へのリクエストを通す設定が追加されました。(図 3.44)



▲図 3.44 ルールが追加された！

3.6.2 今度こそ HTTP でサイトを見てみよう

再び、サーバを立てたときにメモした [パブリック IP アドレス] を、ブラウザで開いてみましょう。(図 3.45)



▲図 3.45 今度こそ HTTP でサイトが見られた！

[Welcome to nginx!] と表示されました！ これでまず、「HTTP でサイトを表示する」はクリアできました。

3.7 ドメイン名の設定をしよう

ウェブサーバを立てて、HTTP でサイトを表示できたので、続いては **HTTPSのサイト用にドメイン名を用意します**。「自分のドメイン名？ そんなの持っていないよ！」という方は、先に「DNS をはじめよう」^{*24}を読んで、ドメイン名を買ってからこの先へ進むようにしてください。^{*25}

それでは、あなたのドメイン名で「ssl.自分のドメイン名」という A レコードを作成し、サーバの [パブリック IP アドレス] と紐付けてください。ネームサーバはお名前.com を使用してもいいですし、AWS の Route53 で設定しても構いません。

なお筆者が「DNS をはじめよう」で購入したのは startdns.fun というドメイン名だったので、ssl.startdns.fun という A レコードを作って、さっそく立てたばかりのサーバの [パブリック IP アドレス] と紐付けます。例えばネームサーバが「お名前.com」なら、DNS 設定の画面でこのように A レコードを追加します。（図 3.46）

*24 <https://mochikoastech.booth.pm/items/881622>

*25 本書で使用する FujiSSL の SSL 証明書は、グローバル IP アドレスに対しては発行できないため、必ずドメイン名が必要です。 <https://www.fujissl.jp/faq/review/1095/>

ホスト名	TYPE	TTL	VALUE				優先	状態	追加
ssl .startdns.fun	A	3600	140	238	.33	.51		有効	追加

▲図 3.46 「ssl. 自分のドメイン名」の A レコードを作成する

A レコードを追加できたかどうかは、次の dig コマンドで確認できます。dig コマンドをたたいた結果、サーバの IP アドレスが返ってくれば A レコードは設定できています。

```
$ dig ssl. 自分のドメイン名 a +short
```

筆者の場合は、次のように表示されました。

```
$ dig ssl.startdns.fun a +short
140.238.33.51
```

ドメイン名が設定できたら、ブラウザでも「<http://ssl. 自分のドメイン名>」をたたいてみましょう。先ほどと同じ NGINX のページが表示されるはずです。(図 3.47)



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

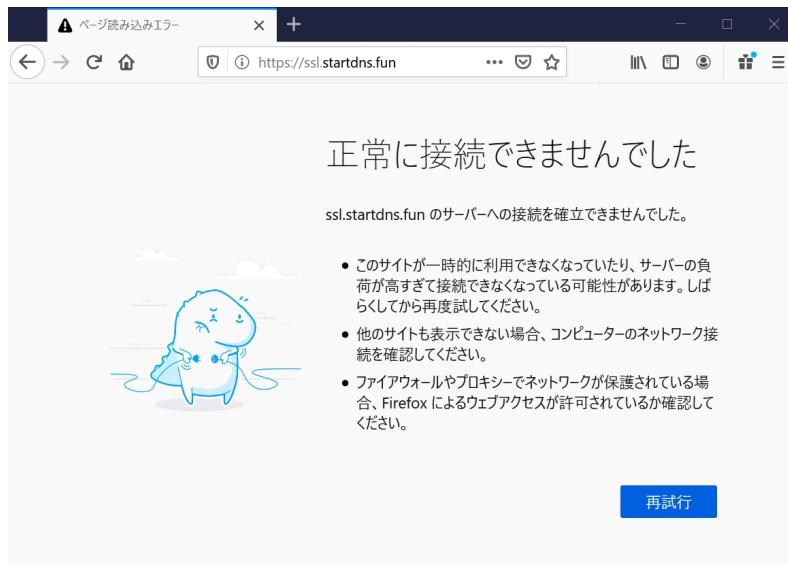
For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

▲図 3.47 「ssl. 自分のドメイン名」で HTTP のサイトが表示された！

HTTP でサイトを見ることができましたね。では同じドメイン名を HTTPS で開いてみるとどうでしょう？ ブラウザで「<https://ssl. 自分のドメイン名>」を開いてみると、「正常に接続できませんでした」と表示（図 3.48）されました。

第3章 サーバを立てて HTTP でサイトを見てみよう



▲図 3.48 HTTPS で開くと [正常に接続できませんでした] と表示された

なぜなら、NGINX で HTTP の設定はしたけれど、HTTPS の設定はまだだからです。それでは HTTPS でもサイトが見られるように、次章で SSL 証明書を取得して、設定を進めていきましょう。

第 4 章

SSL 証明書を取って HTTPS でサ イトを見よう

ウェブサーバとドメイン名を用意したら HTTP でサイトが見られました。

今度は HTTPS でサイトを見るために、SSL 証明書を取得しましょう。

4.1 SSL 証明書にまつわる登場人物

SSL 証明書は、関わってくるファイルが多いので、最初に登場人物をご紹介します。

- 秘密鍵 (startssl.key)
- CSR (startssl.csr)
- SSL 証明書 (ssl_自分のドメイン名.crt)
- 中間 CA 証明書 (ca-bundle.ca)
- SSL 証明書+中間 CA 証明書 (startssl.crt)

次のような流れで SSL 証明書を取得して、HTTPS のサイトを作ります。

1. サーバで秘密鍵を作る
2. 秘密鍵から CSR（証明書署名リクエスト）を作る
3. CSR を認証局に渡して SSL 証明書の発行を依頼
4. TXT レコードを作って DNS 認証
5. メールで SSL 証明書と中間 CA 証明書が届く
6. SSL 証明書と中間 CA 証明書をサーバにアップして 1 ファイルにまとめる
7. NGINX で HTTPS のバーチャルホストを作る

流れも複雑なので、どこに何のファイルを置いたか分からなくなってしまわないよう、
登場人物が集まる場所として、サーバの中で /etc/nginx/ の下に ssl というディレクト
リを作りおきましょう。^{*1}

```
# mkdir /etc/nginx/ssl/  
# cd /etc/nginx/ssl/
```

^{*1}もし `mkdir: cannot create directory '/etc/nginx/ssl': Permission denied` と表示されてしまったら、あなたはいま、うっかり一般ユーザのまま `mkdir` コマンドを実行しています。コマン
ドの例で、左側のプロンプトが「#」のときは、root で実行してください。「`sudo su -`」と書いて Enter
キーを押すと root になります

4.2 密密鍵（startssl.key）を作ろう

最初に作成するのが密密鍵^{*2}です。密密鍵は「.key」や「.pem」^{*3}という拡張子で作成されることが多いです。密密鍵は名前のとおり「密密」にすべきです。つまり、限られた人だけが触れるように管理すべきですので、決してメールに添付して送ったり、サーバ内でどのユーザでも見られる場所に置いたりしてはいけません。

それでは次の openssl コマンドをたたいて、密密鍵を生成しましょう。左から順に「openssl コマンドで／鍵アルゴリズムが RSA の密密鍵を作る／密密鍵は/etc/nginx/ssl/以下に startssl.key というファイル名で保存する／鍵の長さは 2048 ビット」という意味です。できあがった密密鍵は、chmod コマンドで読み書き権限を厳しくしておきます。

```
# openssl genrsa -out /etc/nginx/ssl/startssl.key 2048
Generating RSA private key, 2048 bit long modulus
.....+ ++
e is 65537 (0x10001)

# chmod 600 /etc/nginx/ssl/startssl.key
```

できあがった密密鍵を、cat コマンドで見てみましょう。

```
# cat /etc/nginx/ssl/startssl.key
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAO+s/Gzdb0bzg6QWzCvK5JofMv6izHzlCfMCMhcU7SeBd2tHN
icRA7g5CZq09aaEqv1949cFX5C3bgHx140+epeudrKyUjRwZSpS70mznDBFQByTY
(中略)
InsCw9qu+iZknMKiISw3Krht/898/hq0jqLFJUTbfg9BP8w+JVW4+8hp40Sklymc
NRcvPYUBQy3wK+w527rksodkGZ77c6Q+XxRtH/wpo3H+xwhmJvi+T2o=
-----END RSA PRIVATE KEY-----
```

【コラム】SSL 証明書の密密鍵にパスフレーズは設定すべき？

openssl コマンドで密密鍵を作るとき、-aes128 や-aes256 というオプションを付けると、パスフレーズを聞かれて、指定の暗号方式で暗号化された密密鍵が出力さ

^{*2} さっき作ったのは SSH でログインするための密密鍵で、今から作るのは SSL 証明書を取得するための密密鍵なので別物です

^{*3} 密密鍵が PEM と呼ばれるテキスト形式で生成されることから、pem という拡張子が使われるようす

れます。^{*4} このとき、[Enter pass phrase:] や [Verifying - Enter pass phrase:] と表示された横で、いくらキーボードをたたいて文字を入力しても、黒い画面上は何も表示されません。ですが、ちゃんと文字入力はできているので大丈夫です。入力を間違えたら、Backspace キーで消して書き直すこともできます。

```
# openssl genrsa -aes128 -out /etc/nginx/ssl/with-passphrase.key 2048
Generating RSA private key, 2048 bit long modulus
(中略)
Enter pass phrase: ←秘密鍵のパスフレーズとして「startssl」と入力
Verifying - Enter pass phrase: ←もう一度「startssl」と入力
```

SSH の鍵ペアを作ったときのコラムでも解説しましたが、秘密鍵がパスフレーズで保護されると、秘密鍵を盗んだ誰かが使おうとしても、パスフレーズが分からなければ使えないで安心です。

しかし秘密鍵にパスフレーズが設定されていると、Apache や NGINX といったウェブサーバを再起動した際にも、必ずパスフレーズを聞かれます。もし何かトラブルがあってサーバが OS ごと再起動してしまった場合、折角 NGINX が自動起動する設定になっていても、パスフレーズを聞くところで止まって起動できず、サイトが自動復旧しない、というトラブルが発生します。

このトラブルを回避するには、パスフレーズをファイルに書いておいて、NGINX の自動起動時にそれを読み込むようにする、という方法があります。しかし折角設定したパスフレーズをファイルに書いて同じウェブサーバ内に置いてしまうと、秘密鍵を盗むとき、一緒にパスフレーズのファイルも盗まれてしまう可能性が高くなり、もはやパスフレーズを設定した意味がありません。そのため筆者は、ウェブサーバに設置して使う秘密鍵にはパスフレーズは設定しなくてよい、という考えです。

ちなみに、次のように一度パスフレーズありで作った秘密鍵を、パスフレーズなしで複製することも可能です。

```
# cd /etc/nginx/ssl/
# openssl rsa -in with-passphrase.key -out without-passphrase.key
Enter pass phrase for with-passphrase.key: ←秘密鍵のパスフレーズ
「startssl」を入力
writing RSA key
```

利便性を保つつつ安全性も向上させたければ、本番のウェブサーバで使う秘密

鍵はパスフレーズなし、バックアップとして他のサーバで保管しておく秘密鍵はパスフレーズありにしておく、という方法がいいでしょう。

4.3 CSR (startssl.csr) を作ろう

秘密鍵ができたら、続いて CSR (Certificate Signing Request) の作成に進みましょう。CSR は「.csr」という拡張子で作成されることが多いです。CSR は「Certificate Signing Request (証明書署名リクエスト)」という名前のとおり、認証局に対して「こういう内容で SSL 証明書を作って署名してください」とリクエストする、申請書のようなものです。

次の openssl コマンドを実行します。左から順に、「openssl コマンドで／CSR を作る／新しく作る／元になる秘密鍵は startssl.key ／CSR は startssl.csr というファイル名を作る」という意味です。

```
# cd /etc/nginx/ssl/
# openssl req -new -key startssl.key -out startssl.csr
```

すると、CSR を作成するため、いくつか質問をされます。(表 4.1)*5 コモンネームには、ssl.自分のドメイン名を入力してください。

▼表 4.1 CSR 作成時に聞かれる質問

質問	説明	入力する内容の例	必須/任意
Country Name	国名 (C)	JP	必須
State or Province Name	都道府県名 (S/ST)	Tokyo	必須
Locality Name	市町村名 (L)	Shinjuku-ku	必須
Organization Name	組織名 (O)	mochikoAsTech	必須
Organizational Unit Name	組織単位名 (OU)	入力なし	任意
Common Name	コモンネーム (CN)	ssl.startdns.fun	必須

*4 さっきは-aes128 や-aes256 といった「どの暗号で暗号化するか？」のオプションを何も指定しなかったので、パスフレーズは聞かれず、秘密鍵も暗号化されませんでした

*5 本書では DV 証明書を取得するため「組織単位名 (OU)」は任意としていますが、証明書の種類や認証局によっては必須のところもあるようです。取得時に認証局のサイトで確認しましょう。DV 証明書については後述します

```
Country Name (2 letter code) [XX]:JP
State or Province Name (full name) []:Tokyo
Locality Name (eg, city) [Default City]:Shinjuku-ku
Organization Name (eg, company) [Default Company Ltd]:mochikoAsTech
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:ssl.startdns.fun
```

次の3つは入力不要です。

```
Email Address []:
A challenge password []:
An optional company name []:
```

質問に全て答えると、CSR ができあがります。再び openssl コマンドをたたいて、CSR の中身を見てみましょう。「Subject」と書かれた行を見て、生成された CSR の内容が正しいか確認します。

```
# openssl req -text -in /etc/nginx/ssl/startssl.csr -noout | head -4
Certificate Request:
Data:
Version: 0 (0x0)
Subject: C=JP, ST=Tokyo, L=Shinjuku-ku, O=mochikoAsTech, CN=ssl.startdns.fun
```

内容に問題がなければ、CSR を cat コマンドで表示します。「-----BEGIN CERTIFICATE REQUEST-----」から「-----END CERTIFICATE REQUEST-----」までをコピーしたら、メモ帳にペーストしておきましょう。メモした CSR はこの後使用します。

```
# cat /etc/nginx/ssl/startssl.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICqzCCAZMCAQAwZjELMAkGA1UEBhMCS1AxDjAMBgNVBAgMBVRva3lvMRQwEgYD
VQQHDA1UECgwNbW9jaG1rb0FzVGVjaDEZMBcGA1UE
(中略)
jP1RMQS3PuYDE6QIVJ5zbMC+RIydsQ/ODr9VUHWiYqDPjx+BpphYT5AxMwbw9/m5
noKGvJl1Mt7G03Awa/TX
-----END CERTIFICATE REQUEST-----
```

4.3.1 【ドリル】CSR で入力すべきなのは誰の情報？

問題

A 銀行のウェブサイトを HTTPS で作ることになりました。SSL 証明書^{*6}の取得は A 銀行の代わりに広告代理店の B 社が行い、さらにサイトの制作や運用は A 銀行から Web 制作会社の C 社に委託する場合、CSR で入力する住所や会社名は A 銀行・B 社・C 社のどれにすべきでしょうか？

- A. A 銀行のウェブサイトなんだから A 銀行を入力すべき
- B. A 銀行から任されて SSL 証明書を買うのは B 社だから B 社を入力すべき
- C. 実際にサイトの管理を任せているのは C 社だから C 社を入力すべき

答え _____

解答

正解は A です。ウェブサイトの運営元が A 銀行であることを証明するための SSL 証明書なので、CSR では A 銀行の情報を記載すべきです。

A 銀行のフィッシングサイトが出てきたときに、ユーザが「本物のサイトか確認しよう」と思って証明書の情報を見て、B 社や C 社の情報が表示されたら「A 銀行じゃない！」となってしまいます。

【コラム】openssl コマンドのユーティリティ（道具）たち

第 1 章「SSL/TLS とは？」で説明したとおり、OpenSSL は SSL というプロトコル（約束事）に従って実装された、暗号処理のためのソフトウェアです。OpenSSL のコマンドラインツールである openssl コマンドには、色々なユーティリティ（道具）が詰まっています。秘密鍵を作るとときには `genrsa`、CSR を作るとときには `req` というユーティリティを、openssl コマンドの引数について実行しましたよね。

^{*6} SSL 証明書の種類は「EV 証明書」とします。EV 証明書については第 5 章「SSL/TLS の仕組み」で後述します

```
秘密鍵を作るときは、genrsa というユーティリティを使った  
# openssl genrsa -out /etc/nginx/ssl/startssl.key 2048
```

```
秘密鍵をパスフレーズありからなしにするとときは rsa というユーティリティを使った  
# openssl rsa -in with-passphrase.key -out without-passphrase.key
```

```
CSR を作るときは、req というユーティリティを使った  
# openssl req -new -key startssl.key -out startssl.csr
```

genrsa や req、rsa の他に、どんなユーティリティがあるのか？は、openssl help^{*7}で確認できます。

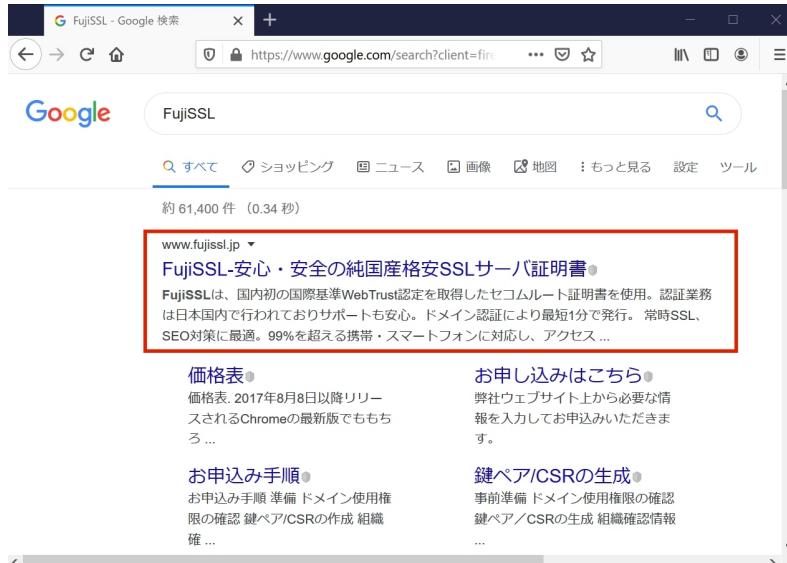
```
$ openssl help  
openssl:Error: 'help' is an invalid command.  
  
Standard commands  
asn1parse      ca          ciphers       cms  
crl           crl2pkcs7   dgst        dh  
dhparam       dsa          dsaparam     ec  
ecparam        enc         engine       errstr  
gendh         gendsa      genpkey     genrsa  
nseq          ocsp        passwd      pkcs12  
pkcs7          pkcs8       pkey        pkeyparam  
pkeyutl       prime       rand        req  
rsa           rsautl      s_client    s_server  
s_time        sess_id     smime      speed  
spkac         ts          verify     version  
x509
```

それぞれのユーティリティの使い方が知りたいときは、man コマンドの引数にユーティリティ名をつけてやれば、マニュアルが表示されます。

```
$ man genrsa
```

4.4 SSL 証明書の取得申請を出そう

それではターミナルは一旦置いておいて、ブラウザを開いてください。ここからはCSRに従って、認証局にSSL証明書を発行してもらいます。[FujiSSL]で検索して、[FujiSSL-安心・安全の純国産格安SSLサーバ証明書]をクリック（図4.1）します。^{*8}



▲図4.1 [FujiSSL-安心・安全の純国産格安SSLサーバ証明書]をクリック

右上の「お申し込みはこちら」をクリック（図4.2）します。

^{*8} 検索すると色々な価格帯のSSL証明書が出てきますが、値段によって何が違うのかについては、第5章「SSL/TLSの仕組み」で後述します。本書ではFujiSSLという安価な証明書を取得します

第4章 SSL証明書を取ってHTTPSでサイトを見よう



▲図4.2 [お申し込みはこちら] をクリック

続いて [お申し込みサイト（ストアフロント）へ] をクリック（図4.3）します。



▲図4.3 [お申し込みサイト（ストアフロント）へ] をクリック

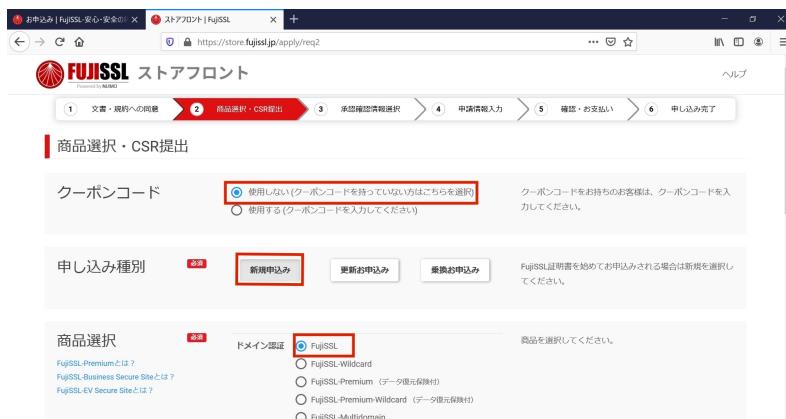
指定された文書と「収集した個人情報の利用目的」を確認した上で、チェックボックスにチェックを入れて [次へ] をクリック（図4.4）します。

4.4 SSL 証明書の取得申請を出そう



▲図 4.4 チェックを入れて [次へ] をクリック

クーポンコードは「使用しない」、申し込み種別は「新規申込み」、商品選択は「ドメイン認証」の「[FujiSSL]」になっていることを確認（図 4.5）します。



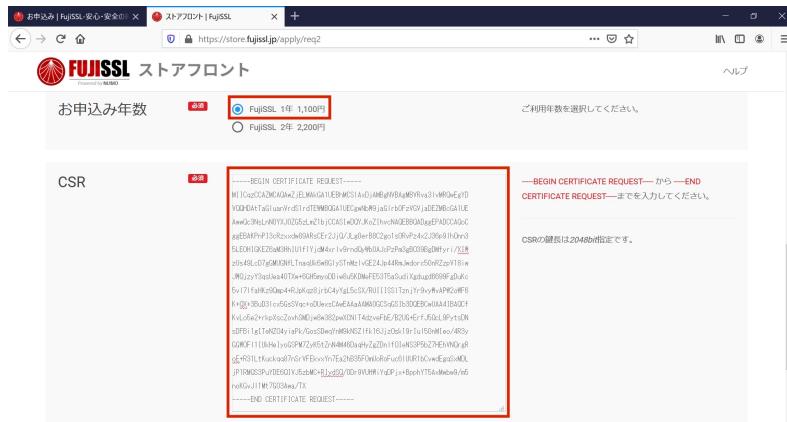
▲図 4.5 クーポンコード、申し込み種別、商品選択を確認

下へスクロールして、お申込み年数が「FujiSSL 1 年 1,100 円】*9 になっていることを確認（図 4.6）したら、先ほどメモ帳にメモしておいた CSR を、「CSR」へ貼り付けます。「-----BEGIN CERTIFICATE REQUEST-----」から「-----END CERTIFICATE

*9 2020 年 2 月現在、FujiSSL の「ドメイン認証シングルタイプ」という SSL 証明書は、有効期間 1 年で税込 1,100 円です

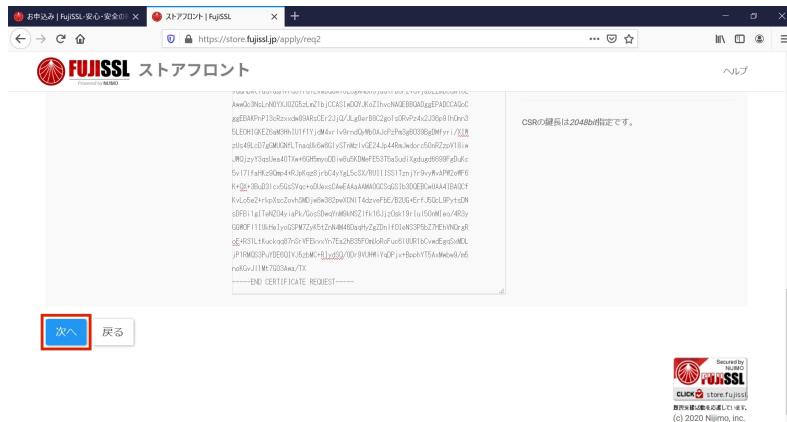
第4章 SSL証明書を取ってHTTPSでサイトを見よう

REQUEST----」まで、丸ごとコピー&ペーストしてください。



▲図4.6 CSRをペーストする

[次へ]をクリック(図4.7)してください。



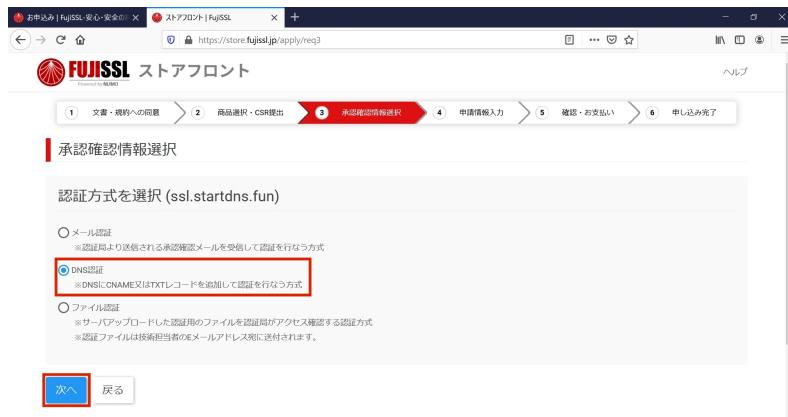
▲図4.7 [次へ]をクリック

【認証方式を選択】へ進みます。これは「CSRのコモンネームで指定したドメイン名が、あなたの持ち物であることをどうやって証明しますか？」ということを聞かれています。対象のドメイン名で、メールを受信してURLを踏む（メール認証）か、指定された内容のリソースレコードをDNSで追加する（DNS認証）か、指定された内容のファイルをウェブサイトにアップする（ファイル認証）か、いずれかの方法で「このドメイン名（筆

4.4 SSL 証明書の取得申請を出そう

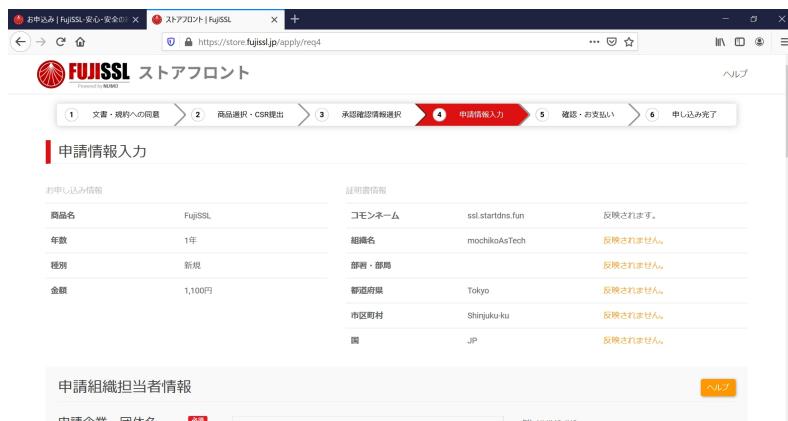
者なら ssl.startdns.fun) は私の持ち物です」ということを証明しなければいけません。

今回は DNS で TXT レコードを追加する、という方法で証明するので、[DNS 認証] を選択（図 4.8）して [次へ] をクリックします。



▲図 4.8 [DNS 認証] を選択して [次へ] をクリック

今回購入するのは DV 証明書です。DV 証明書は「そのドメイン名の使用権があること」だけを証明してくれます。サイトの運営者が日本にいることや、東京の新宿区にオフィスがあること、mochikoAsTech という組織であることなどは確認も証明もしないので、実際の証明書にもそれらの情報は反映されません。（図 4.9）



▲図 4.9 「証明書情報」は「コモンネーム」だけが反映される

[申請組織担当者情報] と [技術担当者情報] を英語表記で入力（図 4.10）します。ここで入力した住所や電話番号、個人名などが外向けに公開されることはありませんので、安心して入力してください。^{*10}

The screenshot shows a web-based application for FujSSL. At the top, there are two tabs: 'お申込み | FujSSL 安心・安全の SSL 証明書' and 'ストアフロント | FujSSL'. The URL in the address bar is https://store.fujssl.jp/apply/req4. The main content area is titled 'FujSSL ストアフロント' and 'From now on'. Below this, there are two sections: '申請組織担当者情報' (Application Organization Contact Information) and '技術担当者情報' (Technical Contact Information). Both sections have red boxes highlighting the input fields for '申請企業・団体名' (Organization Name), 'お名前' (Name), '役職' (Position), '国名' (Country), '郵便番号' (Postal Code), '所在地' (Address), and 'Eメールアドレス' (Email Address).

▲図 4.10 [申請組織担当者情報] と [技術担当者情報] を入力

ここで入力した [E メールアドレス] 宛てに、後で SSL 証明書が送られてきます。メールアドレスを間違えないよう注意してください。すべて入力したら、[次へ] をクリック（図 4.11）します。

^{*10} EV 証明書や OV 証明書の場合は、ここで入力した担当者宛てに実在確認の連絡がります。詳細については第 5 章「SSL/TLS の仕組み」で後述します

4.4 SSL 証明書の取得申請を出そう

The screenshot shows the FujSSL storefront application interface. The address input fields are highlighted with red boxes: '所在地' (Address) contains 'Tokyo' and 'Shinjuku-ku'; '番地' (Address details) contains '4-1-6 Shinjuku' and 'SHINJUKU MIRAINA TOWER'; '電話番号' (Phone number) contains '03-xxxx-xxxx'; and 'Eメールアドレス' (Email address) contains 'startdns.01@gmail.com'. Below the address fields are two buttons: '次へ' (Next) in blue and '戻る' (Back) in grey.

▲図 4.11 「次へ」をクリック

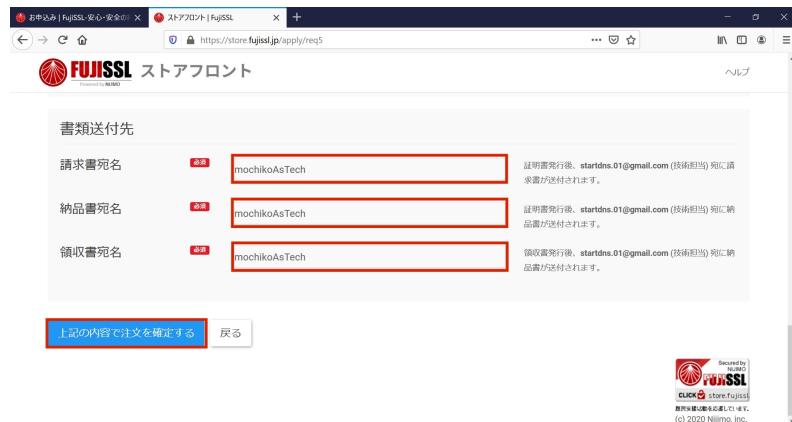
[決済情報入力] に、SSL 証明書代を支払うクレジットカード情報を入力（図 4.12）します。

The screenshot shows the FujSSL storefront application interface for payment information entry. The credit card information fields are highlighted with red boxes: '決済金額' (Amount) is set to '1,100 円' (1,100 yen); 'ご利用いただけるカードの種類' (Card types available) includes JCB, MasterCard, VISA, AMERICAN EXPRESS, DECORIS, and Diners Club; 'クレジットカード番号' (Credit card number) is a masked field; '有効期限' (Expiration date) shows a dropdown for month and year; 'セキュリティコード' (Security code) is a masked field; and 'カード名義' (Cardholder name) is a masked field. Below these fields is a section for '書類送付先' (Document delivery address).

▲図 4.12 クレジットカード情報を入力

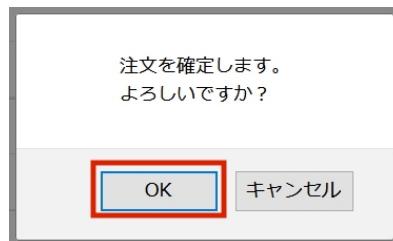
[書類送付先] で [請求書宛名]、[納品書宛名]、[領収書宛名] を記入したら、[上記の内容で注文を確定する] をクリック（図 4.13）します。

第4章 SSL証明書を取ってHTTPSでサイトを見よう



▲図 4.13 [上記の内容で注文を確定する] をクリック

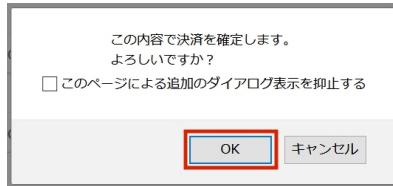
「注文を確定します。よろしいですか？」と表示（図 4.14）されたら、[OK] をクリックします。



▲図 4.14 [OK] をクリック

「この内容で決済を確定します。よろしいですか？」と表示（図 4.15）されます。「1年間有効な SSL 証明書を 1,100 円で買うんだ！」というケツイ^{*11}をしたら、[OK] をクリックします。

^{*11} UNDERTALE というゲームは、ゲームオーバーになるたび「ケツイを ちからに かえるんだ…！」というメッセージが表示されます。筆者はパビルスが好きですが、初見でうっかり殺してしまい、同僚に人でなし扱いされました



▲図 4.15 1,100 円払うケツイをして [OK] をクリック

[お申込み完了] のページが表示（図 4.16）されました。

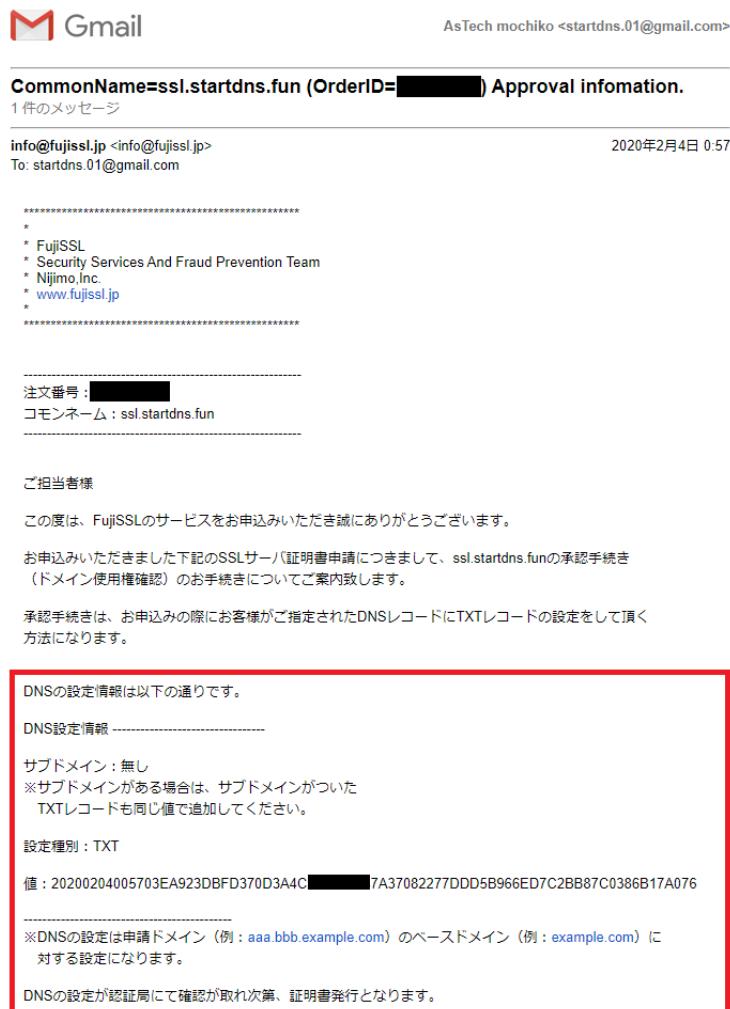


▲図 4.16 SSL 証明書の申し込みが完了した！

先ほど登録したメールアドレス宛に、DNS 設定情報を知らせるメールが届いていますので確認しましょう。

4.5 DNS の設定をしよう

[CommonName=ssl.自分のドメイン名] から始まる件名のメール（図 4.17）がすぐに届きます。例えば CSR のコモンネームで指定したドメイン名が [ssl.startdns.fun] だった場合、ベースドメイン名である [startdns.fun] の TXT レコードを追加して、メールに書いてある値を設定するよう書いてあります。



▲図 4.17 追加すべき TXT レコードの値がメールで届いた

例えば、そのドメイン名で使用しているネームサーバが「お名前.com」なら、DNS 設定の画面で次のように TXT レコードを追加（図 4.18）します。今回はサブドメインを含まないドメイン名を追加するので、[ホスト名] には何も入力しません。[VALUE] には、メールに書いてあった値をそのままコピー＆ペーストします。

4.6 SSL 証明書と中間 CA 証明書をメールで受け取ろう

A/AAAA/CNAME/MX/NS/TXT/SRV/DS/CAAレコード						
ホスト名	TYPE	TTL	VALUE	優先	状態	追加
.startdns.fun	TXT ▾	3600	20200204005703EA923DBF	有効 ▾	有効	<input type="button" value="追加"/>

▲図 4.18 「自分のドメイン名」の TXT レコードを追加する

ちょっとだけ時間をおいてから、TXT レコードが追加されたか確認してみましょう。TXT レコードが追加されたかどうかは、次の dig コマンドで確認できます。dig コマンドをたたいた結果、メールに書いてあった値が返ってくれば TXT レコードは設定できています。

```
$ dig 自分のドメイン名 txt +short
```

筆者の場合は、次のように表示されました。

```
$ dig startdns.fun txt +short  
"20200204005703EA923DBFD3 (中略) DD5B966ED7C2BB87C0386B17A076"
```

これで TXT レコードの設定^{*12}は完了です。

4.6 SSL 証明書と中間 CA 証明書をメールで受け取ろう

TXT レコードを追加してから、おおよそ 30 分後に SSL 証明書がメール（図 4.19）で届きました。

^{*12} いま設定した TXT レコードの他に、CAA レコードが SSL 証明書の発行に関わってくることがあります。CAA レコードについては、「DNS をはじめよう」の「〈トラブル〉 CAA レコードが原因で SSL 証明書が発行できなかった」で説明しています



▲図 4.19 SSL 証明書がメールで届いた

メールに添付されている ZIP ファイル (ssl.自分のドメイン名.zip) に SSL 証明書が入っていますのでダウンロードします。(図 4.20)



▲図 4.20 メールに添付されている ZIP ファイルをダウンロード

Windows の方も Mac の方も、ダウンロードした ZIP ファイルはデスクトップに置いてください。(図 4.21)



▲図 4.21 ダウンロードした ZIP ファイルはデスクトップに置く

ZIP ファイルを右クリックして、[すべて展開] をクリックします。(図 4.22)

4.6 SSL 証明書と中間 CA 証明書をメールで受け取ろう



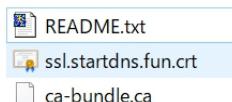
▲図 4.22 右クリックして [すべて展開] をクリック

[展開] をクリックします。(図 4.23)



▲図 4.23 [展開] をクリック

展開したフォルダ（図 4.24）の中の [ssl. 自分のドメイン名.crt] が SSL 証明書で、[ca-bundle.ca] が中間 CA 証明書です。README.txt はファイルの説明書です。



▲図 4.24 ssl. 自分のドメイン名.crt が SSL 証明書で、ca-bundle.ca が中間 CA 証明書

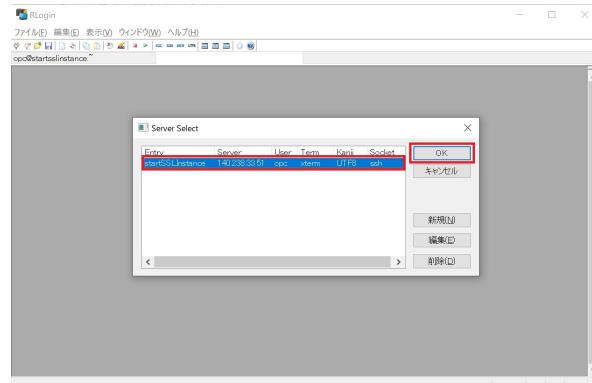
どちらも必要なものなので、これから SSL 証明書と中間 CA 証明書の 2 つをサーバにアップロードしましょう。

4.7 SSL証明書をサーバに設置しよう

それではSSL証明書をサーバに設置します。NGINXではSSL証明書と中間CA証明書の2つを、1ファイルにまとめて使用するので、まずはアップロードして、それから1ファイルにまとめる作業を行ないます。Windows、Macの順番で手順を説明します。

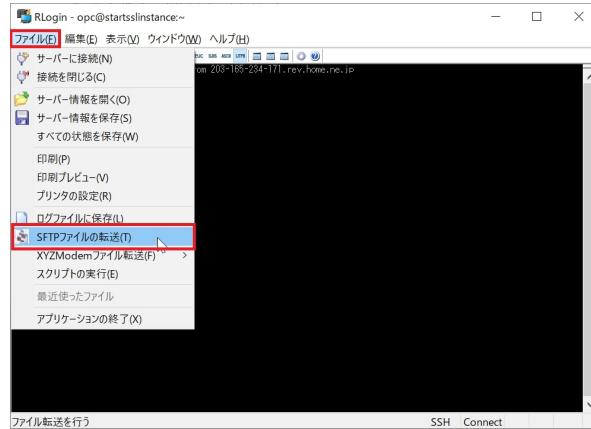
4.7.1 WindowsでSSL証明書と中間CA証明書をアップロードしよう

Windowsのパソコンを使っている方は、RLoginを起動します。[startSSLInstance]を選択して、[OK]をクリック（図4.25）します。



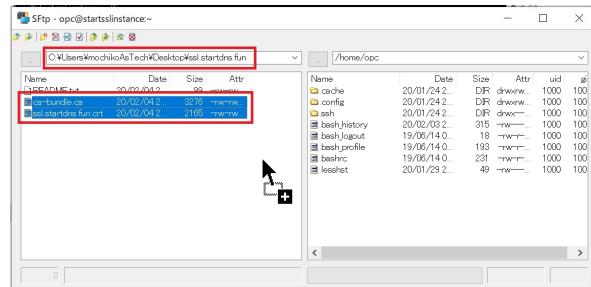
▲図4.25 RLoginを起動して [startSSLInstance] を選択してログイン

黒い画面が開いたら、[ファイル]から[SFTPファイルの転送]をクリック（図4.26）します。



▲図 4.26 「[ファイル]」から「[SFTP ファイルの転送]」をクリック

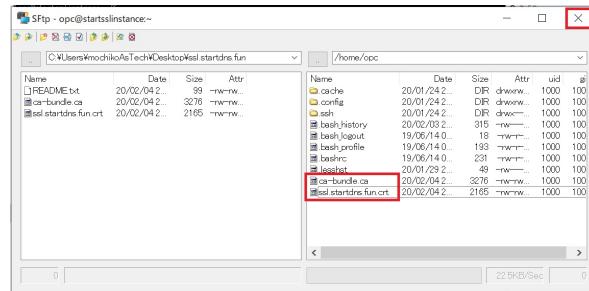
左側にあなたのパソコンのファイルが表示されていて、右側にサーバのファイルが表示されています。左側で、デスクトップに展開したフォルダ^{*13}を選択したら、「ssl. 自分のドメイン名.crt」と「ca-bundle.ca」を右側にドラッグ＆ドロップ（図 4.27）してください。これでサーバの「/home/opc」にファイルがアップロードされます。



▲図 4.27 「ssl. 自分のドメイン名.crt」と「ca-bundle.ca」を右側にドラッグ＆ドロップ

右側のサーバに「ssl. 自分のドメイン名.crt」と「ca-bundle.ca」がアップロードされたら、×を押してファイル転送の画面は閉じて構いません。（図 4.28）

^{*13} 筆者の場合は「C:\Users\mochikoAsTech\Desktop\ssl.startdns.fun」でした



▲図 4.28 [ssl. 自分のドメイン名.crt] と [ca-bundle.ca] がサーバにアップされた

4.7.2 Mac で SSL 証明書と中間 CA 証明書をアップしよう

Mac を使っている方は、ターミナルを起動してください。ssh コマンドではなく、scp コマンドを使って、Mac の中にある [ssl. 自分のドメイン名.crt] と [ca-bundle.ca] を、サーバへアップロードします。[ZIP を展開したフォルダ]、[ssl. 自分のドメイン名.crt]、[パブリック IP アドレス] の部分は、ご自身のものに書き換えてください。

```
$ cd ~/Desktop/ZIP を展開したフォルダ
$ scp -i ~/Desktop/startSSLKey ssl. 自分のドメイン名.crt ca-bundle.ca opc@パブリック IP アドレス:/home/opc/
ssl. 自分のドメイン名.crt      100%  0    0.0KB/s   00:00
ca-bundle.ca                   100%  0    0.0KB/s   00:00
```

4.7.3 証明書を 1 ファイル (startssl.crt) にまとめよう

これで Windows の方も、Mac の方も、サーバに SSL 証明書と中間 CA 証明書をアップロードできました。それではターミナルでサーバにログインして、先ほどアップロードした [ssl. 自分のドメイン名.crt] と [ca-bundle.ca] が、ちゃんと「/home/opc/」以下に存在していることを確認しましょう。

```
$ sudo su -
# ls /home/opc/
ca-bundle.ca  ssl. 自分のドメイン名.crt
```

続いて 2 つのファイルから、新たに [startssl.crt] というファイルを作りましょう。^{*14}NGINX では SSL 証明書と中間 CA 証明書という 2 つのファイルを、1 つのファイルにがっちゃんこ！ とつなげて使うためです。

```
# cd /etc/nginx/ssl/  
# awk 1 /home/opc/ssl_自分のドメイン名.crt /home/opc/ca-bundle.ca > startssl.crt
```

2 つのファイルを 1 つにつなげたら、cat コマンドで [startssl.crt] を確認してみましょう。次のように「-----BEGIN CERTIFICATE-----」と「-----END CERTIFICATE-----」が、繰り返し 3 つ表示されれば大丈夫^{*15}です。

```
# cat /etc/nginx/ssl/startssl.crt  
-----BEGIN CERTIFICATE-----  
MIIF/DCCBOSgAwIBAgIQaoS/P1b4mCR8mn5/WUrI6zANBgkqhkiG9w0BAQsFADBn  
MQswCQYDVQQGEwJKUDElMCMGA1UEChMcUOVDT0QgVHJ1c3QgU3lzdGVtcyBDTy4s  
(中略)  
31pTIUPabkFxDPjs1Vw9c7z3Vgk2fnpuwE21rE+46zrJ3oTRqsABDbYreK1a5vsG  
tcnpU1L1hrk/rC3JuI2ttHVaHU+1JCgTSRpvO3a44azDy15T5C97dGxuowPgcaMQ  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
MIIEpzCCA4+gAwIBAgIJIrnxVPM8X14AMA0GCSqGSIB3DQEBCwUAMFOxCzAJBgNV  
BAYTAkpQMWSUwIwYDVQQKExxTRUNPTSBUcnVzdCETeXNOZW1zIENPLixMVEQuMScw  
(中略)  
g3tiJAlFIpfXXD4cArZo6Z1XJ26B4H7vk5GmyR6poDy/CRvC7VIz3xp6o2348W1j  
32S9pEuZhTxtMvPjnsHiWPNdz8pHv21x7bYwDnocwN2uk3QrrljxtTQ9evg==  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
MIIEcjCCA1qgAwIBAgIJErnw+nLg2EjGMAOGCSqGSIB3DQEBCwUAMFAxCzAJBgNV  
BAYTAkpQMrgwFgYDVQQKEw9TRUNPTSBUcnVzdC5uZXQxJzA1BgNVBAstHlN1Y3Vy  
(中略)  
u5ZuCjxerxj3qS1rM46bcEfjopnaD7hnJXSYiL1d0yw5zSW2PEe+LHdoIAb2I6D8  
8UFJHOCl16sY518jhjk00s1yeu1C/RcY0+NBNHKZkFEeEb6ezOsg=  
-----END CERTIFICATE-----
```

これで証明書ファイルの準備は完了です。

^{*14} このとき cat コマンドで結合すると、SSL 証明書と中間 CA 証明書の間に改行が入らず、「-----END CERTIFICATE----------BEGIN CERTIFICATE-----」のようになってしまふため、awk コマンドを使います

^{*15} 1 つのファイルにつなげるときには、SSL 証明書が上で、中間 CA 証明書が下です。順番には意味があるので、逆にならないよう注意してください

4.8 NGINXでHTTPSのバーチャルホストを作ろう

「/etc/nginx/conf.d/」というディレクトリの下に移動して、もともとあった設定ファイルをmvコマンドでバックアップしておきます。

```
# cd /etc/nginx/conf.d/  
# mv default.conf default.conf.backup
```

viコマンドで、同じ場所に新しい設定ファイル^{*16}を作ります。viコマンドでファイルを編集するときは、i(アイ)を押してから入力します。書き終わったらESCキーを押して、「:wq」と入力してEnterキーを押せば変更が保存されます。ssl_ciphersは、紙面上では途中で改行して4行になっていますが、実際は改行なしで1行です。

```
# vi startssl.conf  
  
server {  
    listen 80 default_server;  
    return 301 https://$host$request_uri;  
}  
  
server {  
    listen      443 ssl http2;  
    server_name ssl.startdns.fun;  
  
    # 密密鍵  
    ssl_certificate_key /etc/nginx/ssl/startssl.key;  
    # SSL証明書+中間CA証明書  
    ssl_certificate      /etc/nginx/ssl/startssl.crt;  
  
    # 暗号スイート ↓ 実際は改行なしで1行  
    ssl_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:  
    ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:  
    ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:  
    DHE-RSA-AES128-GCM-SHA256;  
  
    # プロトコルバージョン  
    ssl_protocols      TLSv1.2;  
    # 暗号スイートの順序はサーバが決める  
    ssl_prefer_server_ciphers  on;  
  
    location / {  
        root   /usr/share/nginx/html;
```

^{*16} 設定ファイル(startssl.conf)を手入力するのがしんどい場合は、PDF版からコピー&ペーストしても構いません。紙の本を購入された方は、PDF版を無料でダウンロードできます。ダウンロード方法は「あとがき」をご確認ください

```
        index  index.html index.htm;  
    }  
}
```

ちなみに暗号スイートとは、「認証は RSA で、鍵交換は ECDHE、暗号化アルゴリズムは AES を使う」のように、セキュリティの各パラメータをセットにしたものです。ファーストフード店の「チキンクリームポットパイ+ポテト+コーラ S のセット」や「オリジナルチキン+ビスケット+アイスティー S のセット」みたいなものだと思ってください。どんなセットがあるのか、一覧は IANA のページ^{*17}で確認できます。

まずはクライアント（ブラウザ）が、希望する暗号スイート群をサーバに送ります。クライアントの希望を受け取ったサーバは、ssl_ciphers ディレクティブに書いてある暗号スイートを上から順に探していく、クライアントの希望と共に通するものが見つかれば、その暗号スイートを使用します。つまりクライアントが「フィレオフィッシュ+ポテト+烏龍茶のセット！ なければキッズナゲット+ビスケット+烏龍茶のセット！」と希望しても、サーバ側の ssl_ciphers ディレクティブにそのセットがなければ、接続できずに【クライアントとサーバーで、共通の SSL プロトコル バージョンまたは暗号スイートがサポートされていません。】と表示されてしまうのです。（図 4.29）



▲図 4.29 共通の暗号スイートが見つからず、接続できなかった様子

そのため ssl_ciphers ディレクティブに、セキュリティを重視して強度の高い暗号スイートだけを書くのか、それとも古い端末との互換性を重視して脆弱な暗号スイートも含めて書くのかは、安全性と対応端末の多さのバランスを考えて決めていくことになります。

^{*17} Transport Layer Security (TLS) Parameters <https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml>

す。本来はサイトの運営者が自分で考えて判断すべきことではあります、次のように設定値の参考を得ることもできます。

- IPA^{*18}が公開している「SSL/TLS 暗号設定ガイドライン^{*19}」では、「高セキュリティ型」「推奨セキュリティ型」「セキュリティ例外型」として3つのssl_ciphersが提示されています。
- Mozilla^{*20}が提供する「Mozilla SSL Configuration Generator^{*21}」というサイト(図4.30)を使うと、NGINXとopensslのバージョンを指定して、「Modern」「Intermediate」「Old」を選ぶだけで、適した設定ファイルが出力されます。

The screenshot shows the Mozilla SSL Configuration Generator interface. It has three main sections: 'Server Software' (Apache, AWS ALB, AWS ELB, Caddy, Dovecot, Exim, Golang, HAProxy, lighttpd), 'Mozilla Configuration' (Modern, Intermediate, Old), and 'Environment' (Server Version: 1.17.7, OpenSSL Version: 1.1.1d). Under 'Miscellaneous', there are checkboxes for 'HTTP Strict Transport Security' (which also redirects to HTTPS if possible) and 'OCSP Stapling'. Below the configuration form, the generated configuration file is displayed:

```
# generated 2020-02-10, Mozilla Guidelines v5.4, nginx 1.17.7, OpenSSL 1.1.1d, intermediate configuration
# https://ssl-config.mozilla.org/server-nginx&version=1.17.7&config=intermediate&openssl=1.1.1.10&guideline=5.4
server {
```

▲図4.30 環境や設定を選ぶだけで適した設定ファイルが出力されるMozillaのサイト

設定ファイルが書けたら、構文エラーがないかテストをします。もし書き間違いがあれば、ここでエラーメッセージとして表示されます。

```
# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

[test is successful]と表示されたら、NGINXを再起動して設定を反映しましょう。

*18 情報処理推進機構のこと。情報処理技術者試験を実施しているのもIPAです

*19 本書では「高セキュリティ型の設定例（楕円曲線暗号あり）」を使用しています。 https://www.ipa.go.jp/security/vuln/ssl_crypt_config.html

*20 ブラウザのFirefoxを作っているあのMozillaです

*21 <https://ssl-config.mozilla.org/>

```
# systemctl restart nginx.service
```

それから、サイトへアクセスしたときに表示される、index.html のメッセージも [Let's start SSL/TLS] に変えておきましょう。

```
# cd /usr/share/nginx/html/
# cp -p index.html index.html.backup
# echo "Let's start SSL/TLS" > index.html
```

4.9 HTTPS でサイトを開いてみよう

証明書を設置して、NGINX の設定ファイルを修正し、NGINX の再起動もしました。それではブラウザで「<https://ssl.startdns.fun>」を開いてみましょう。鍵マーク付きで HTTPS のページが表示されるはずです。(図 4.31)



▲図 4.31 HTTPS でサイトが表示された！

おめでとうございます！ これでサイトを HTTPS にできました！

【コラム】ロードバランサでも SSL ターミネーションできる

なお本書ではウェブサーバに SSL 証明書を設置しましたが、ウェブサーバの手前にロードバランサを置いて、そこに SSL 証明書を設置する「SSL ターミネーション」という方法もあります。その場合、HTTPS で通信するのは、ユーザのパソコンから終端となるロードバランサーまでで、ロードバランサーとウェブサーバの間は HTTP で通信するのが一般的です。

この方法には、次のようなメリットがあります。

- 暗号化や復号の処理を行なう終端がロードバランサになるので、ウェブサー

バの処理負荷が下がる

- SSL 証明書を設置するのはロードバランサの一箇所だけで済む

今回は取得も設置も手作業で行ないましたが、ウェブサーバが負荷分散のために何十台もあるような環境で、1台1台にSSL証明書とCA証明書を設置^{*22}してNGINXを再起動していくのは大変です。手前のロードバランサでSSLターミネーションするやり方なら、その先にウェブサーバが何台であろうと、SSL証明書を設置するのはロードバランサーの1箇所だけで済みます。

さらにAWSのELB^{*23}とACM^{*24}を組み合わせて使うと、今回手作業で行なった秘密鍵とCSRの生成や、SSL証明書と中間CA証明書の設置なども、まるごと任せられます。これは手作業で3時間かけて皮から餃子を作つて食べるのと、お店で餃子定食を頼んで食べるのくらい、手間暇に差があるので、本書では「餃子を作る工程を一通り体験すること」を目的として、敢えて手作業でSSL証明書を取得しました。

^{*22} ちなみに本書で扱ったFujiSSLでは、取得した1枚のSSL証明書を複数台のウェブサーバに設置しても構いません。ですが認証局やSSL証明書によっては、「1台にしか設置できません。2台以上に設置する場合は台数分だけ購入してください」というケースもありますので、購入前にライセンス形態を確認しておきましょう

^{*23} Elastic Load Balancingの略。AWSのロードバランサ

^{*24} AWS Certificate Managerの略。SSL証明書の取得や設置、管理をまるごとやってくれるAWSのサービス

第5章

SSL/TLS の仕組み

SSL 証明書を取得して、実際に HTTPS のサイトを公開できました。

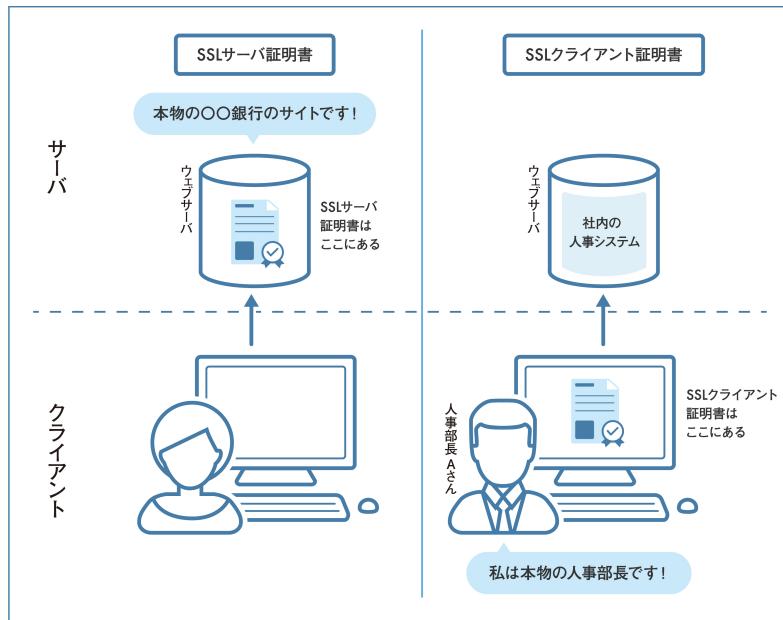
実践したことと照らし合わせながら、SSL/TLS の仕組みを学んでいきましょう。

5.1 SSL 証明書とは

サイトを HTTPS 化するために SSL 証明書を取得しました。でも SSL 証明書とは、結局のところ何で、どういう働きをしているのでしょうか？

5.1.1 SSL サーバ証明書と SSL クライアント証明書

皆さんのがさっそく取得した「SSL 証明書」ですが、実は SSL サーバ証明書と SSL クライアント証明書の 2 種類（図 5.1）があります。ざっくり言うとサーバの身元を証明するのが SSL サーバ証明書で、クライアントの身元を証明するのが SSL クライアント証明書です。



▲図 5.1 SSL サーバ証明書と SSL クライアント証明書

単に「SSL 証明書」という略称で呼んだ場合は、「SSL サーバ証明書」のことを指す場合がほとんどです。本書でも SSL サーバ証明書のことを指して、SSL 証明書という言葉を使用しています。

5.1.2 SSL 証明書はどんなときに使われている？

そんな SSL 証明書は、どんなときに使われているのでしょうか？ SSL 証明書は、あなたがブラウザで「`https://`」から始まる URL のサイトを開いて、次のようなマーク（図 5.2、図 5.3）が表示されているときに使われています。

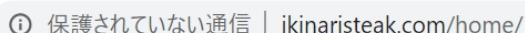


▲図 5.2 Chrome で HTTPS のサイトを開いたとき



▲図 5.3 Firefox で HTTPS のサイトを開いたとき

逆に「`http://`」から始まる URL のサイトを開いて、次のようなマーク（図 5.4、図 5.5）が表示されているときは使われていません。



▲図 5.4 Chrome で HTTP のサイトを開いたとき



▲図 5.5 Firefox で HTTP のサイトを開いたとき

5.2 SSL 証明書は異なる 3 つの仕事をしている

「SSL 証明書は `https://` から始まるページで使われている」ということが分かりました。では `https://` から始まるページにおいて、SSL 証明書は一体何をしているのでしょうか？

うか。

実は、SSL 証明書は異なる 3 つの仕事をしています。例えば、あなたが銀行振込をするためにイグザンブル銀行のウェブサイト (<https://bank.example.com/>) をブラウザで開いたとき、SSL 証明書は次のような 3 つの仕事をします。

- **なりすましを防ぐ**

- 「bank.example.com さん、ページを見て」というリクエストに対して、レスポンスでページを返してきたのが、本当に bank.example.com であることを認証する

- **データの改ざんを防ぐ**

- イグザンブル銀行のウェブサイトで入力した振込先の情報が、サーバに届くまでに他の振込先に書き換えられていないことを確認する
- リクエストに対して返ってきた残高のページが、クライアントまでの途中経路で改ざんされていないことを確認する

- **情報の漏洩を防ぐ**

- サーバへ送信した ID やパスワード、振込先の情報などが第三者に見られないよう暗号化して保護する
- 入出金の明細がサーバからクライアントへ届くまでの間に、第三者に見られないよう暗号化して保護する

厳密には、3 つめの暗号化は SSL 証明書そのものの働きではありません。暗号化に必要な鍵交換を行なう過程で SSL 証明書が仕事をしています。

5.2.1 HTTPS の実際の流れ

では HTTPS の全体を流れを掴むため、あなたが作ったサイト (<https://ssl.自分のドメイン名/>) をブラウザで開いたときのやりとりを、ざっくりと追いかけてみましょう。あなたのブラウザが「クライアント」で、Oracle Cloud 上で立てた HTTPS のサイトが動いているのが「サーバ」です。

認証を行なう

先に RSA^{*1}による認証を行ないます。

1. ブラウザで HTTPS のサイト (<https://ssl.自分のドメイン名/>) を開く

^{*1} もともとは認証だけでなく鍵交換も RSA で行なわれていましたが、RSA による鍵交換は、一度秘密鍵が盗まれてしまうと、過去のやりとりもさかのぼって全ての暗号データを復号可能という問題がありました。そのため TLS1.3 では RSA 鍵交換は廃止されています

2. クライアントから Oracle Cloud のウェブサーバへリクエストを投げる
3. サーバがリクエストを受け取る
4. サーバが、サーバ内にある FujiSSL の SSL 証明書をレスポンスで返す
 - この「SSL 証明書」は次の 2 つを指す
 - SSL 証明書本体（公開鍵を含む）
 - 認証局による署名（SSL 証明書本体のハッシュ値を認証局の秘密鍵で暗号化^{*2}したもの）
5. クライアント側で SSL 証明書を受け取って次の 5 つを行なう
6. ブラウザのトラストアンカー（信頼する証明書）に含まれている認証局によって発行された SSL 証明書なのか確認
 - これで渡された SSL 証明書を信頼してよいことが分かる
7. 認証局による署名を、ブラウザのトラストアンカー（信頼する証明書）に含まれる公開鍵で復号して、証明書本体のハッシュ値を取り出す
8. ハッシュ関数で SSL 証明書本体のハッシュ値を出力
9. 取り出したハッシュ値と、自分で出力したハッシュ値を突き合わせて同一であることを確認する
 - これで渡された SSL 証明書本体が改ざんされていないことが分かる
10. SSL 証明書本体の SAN^{*3}に記載されている FQDN^{*4}と、リクエスト先の FQDN (ssl.自分のドメイン名) が同一であることを確認
 - これでレスポンスを返してきた相手がなりすましてないことが分かる

DH 鍵交換による暗号化通信を行なう

認証が終わると、続けて DH 鍵交換^{*5}を行ないます。

^{*2} ここは実態としては「秘密鍵で復号」らしいのですが、まだ暗号化していないものを復号するイメージを筆者はうまく理解できなかったので、暗号化としています。詳しくはこちらを参照ください。「電子署名=『秘密鍵で暗号化』」という良くある誤解の話 - Qiita https://qiita.com/angel_p_57/items/d7ffb9ec13b4dde3357d

^{*3} Subject Alternative Name の略。どのドメイン名に対する SSL 証明書なのか、対象のドメイン名が書いてあるフィールド

^{*4} FQDN は Fully Qualified Domain Name の略で、日本語だと完全修飾ドメイン名。example.co.jp というドメイン名があったとき、個々の example や co や jp や example.co などは相対ドメイン名で、example.co.jp が FQDN です

^{*5} Diffie-Hellman 鍵交換

1. サーバは DH 公開鍵交換のために使い捨ての鍵ペア（秘密鍵・公開鍵）を作る
 - この鍵ペアは、SSL 証明書を取得するときに作った鍵ペア（秘密鍵・公開鍵）とは別物
 - 以後この鍵ペアを「サーバの DH 用の秘密鍵・公開鍵」と呼ぶ
 2. クライアントも DH 公開鍵交換のために使い捨ての鍵ペア（秘密鍵・公開鍵）を作る
 - 以後この鍵ペアを「クライアントの DH 用の秘密鍵・公開鍵」と呼ぶ
 3. サーバは SSL 証明書用の秘密鍵で、サーバの DH 用の公開鍵を暗号化してクライアント側に渡す
 4. クライアントは「認証」で受け取った SSL 証明書本体に含まれていた「SSL 証明書用の公開鍵」で、サーバの DH 用の公開鍵を復号する
 5. クライアントは自分が作った DH 用の公開鍵を、サーバの DH 用の公開鍵で暗号化してサーバに送る
 6. サーバは、受け取ったクライアントの DH 用の公開鍵を、サーバの DH 用の秘密鍵で復号する
 7. これでクライアントとサーバは、お互いに相手の DH 用の公開鍵を持っている状態になる
 - サーバの DH 用の公開鍵と、クライアントの DH 用の公開鍵のセットを、共通鍵の素（プリマスターシークレット）と呼ぶ
 8. クライアントとサーバは、それぞれプリマスターシークレットを素にして共通鍵を作る
 9. 以降、同一セッションの間は、双方この共通鍵で暗号化および復号してデータをやりとりする
- このように RSA による認証と、DH 鍵交換の組み合わせで、HTTPS の暗号化通信が行なわれています。証明書の鍵ペア（秘密鍵・公開鍵）だけでなく、こんなに色々な種類の鍵ペアが関わっていたんですね。

5.2.2 正当性を証明する中間 CA 証明書

ではここで、サイバートラストが提供する「SSL 証明書の設定確認ツール」^{*6}を使って、あなたが作ったサイトの SSL 証明書を確認（図 5.6）してみましょう。[FQDN1] に [ssl.

^{*6} https://sstool.cybertrust.ne.jp/support_tool/index01.php

5.2 SSL 証明書は異なる 3 つの仕事をしている

自分のドメイン名] を入力し、[設定を確認する] をクリックします。



▲図 5.6 [ssl. 自分のドメイン名] を入力して [設定を確認する] をクリック

すると「証明書は正しく設定されています。」というメッセージと共に、証明書の階層が表示（図 5.7）されました。[中間 CA 証明書 1] と [中間 CA 証明書 2] で、それぞれ [詳細情報を表示する] をクリックすると、詳細が表示されます。

第5章 SSL/TLS の仕組み



▲図 5.7 証明書の階層が表示された

いろいろなことが記載されていますが、要約すると次のことが書かれています。

- SSL サーバ証明書

- [ssl.startdns.fun] というサイトの運営者が、確かにそのドメイン名を管轄していることを、[SECOM Trust Systems CO.,LTD.] が証明する
- 中間 CA 証明書 1
 - SSL サーバ証明書を発行した [SECOM Trust Systems CO.,LTD.] が正当な認証局であることを、[Security Communication RootCA2] が証明する
- 中間 CA 証明書 2
 - 中間 CA 証明書 1 を発行した [Security Communication RootCA2] が正当な認証局であることを、[Security Communication RootCA1] が証明する

信頼の連鎖ですね。[ssl.startdns.fun] というサイトの運営者を仮に A さんとすると、A さんが [ssl.startdns.fun] というドメイン名の持ち主であることを B さんが、B さんの正当性を C さんが、C さんの正当性を D さんが証明しています。

SSL 証明書と中間 CA 証明書 1、2 は [startssl.crt] というファイル名でサーバに設置されており、HTTPS でサイトを開いたときにサーバからクライアント（ブラウザ）へ渡されました。

ところであなたは見も知らぬ D さんが「C さんの正当性を証明する！」と言ったところで信頼できますか？ D さんこと [Security Communication RootCA1] の正当性は、いったい誰が証明するのでしょうか？

5.2.3 ルート証明書はトラストストアにある

実は、中間 CA 証明書 2 を発行した D さんこと [Security Communication RootCA1] の正当性を証明する「ルート証明書」は、皆さんのが使っているブラウザに最初から入っています。Chrome で [設定] を開いたら [証明書] で検索（図 5.8）して、[証明書の管理] をクリックしてみましょう。



▲図 5.8 [証明書の管理] をクリック

[信頼されたルート認証機関] のタブをクリック（図 5.9）すると、そこに D さんこと

[Security Communication RootCA1] の正当性を証明する「ルート証明書」がありました。ちなみにルート証明書の「発行先」と「発行者」を見ると、どちらも同じ [Security Communication RootCA1] になっています。これは D さんの正当性は D さん自身が証明し、ブラウザが D さんのルート証明書を「信頼できるルート証明書である」と判断して格納場所（ルートトラストストア）に入れているので、信頼の連鎖が繋がった、ということです。



▲図 5.9 「信頼されたルート認証機関」にルート証明書があった

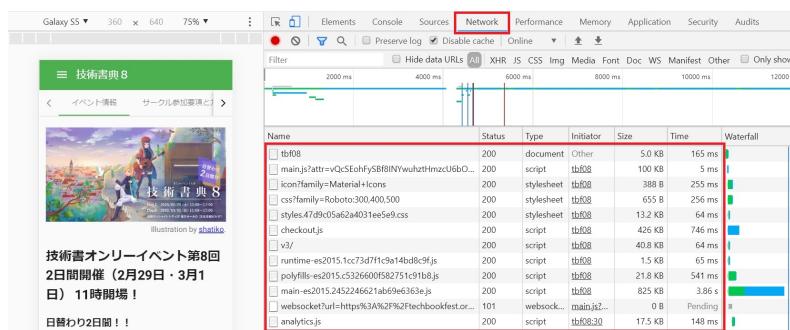
このように SSL 証明書と中間 CA 証明書がウェブサーバにあり、信頼の起点となるルート証明書がブラウザにあることで、SSL 証明書は成り立っています。SSL 証明書ってこんな仕組みになっていたんですね。

5.2.4 ウェブページは1往復で表示されるわけじゃない

ところで1つのウェブページを表示するとき、クライアント（あなたのブラウザ）とサーバのやりとりは、「トップページをください」「はい完成品をどうぞ」の一往復だけではありません。次のように、何往復ものリクエストとレスポンスでページを揃えていくてページが表示されます。

- ・「techbookfest.org さん、 tbf08 のページをください」「はい、 HTML をどうぞ」
- ・「techbookfest.org さん、 main.js をください」「はい、 main.js をどうぞ」
- ・「techbookfest.org さん、 styles.css をください」「はい、 styles.css をどうぞ」
- ・「techbookfest.org さん、 top.jpg をください」「はい、 top.jpg をどうぞ」

Chrome のメニューから [その他のツール] の [デベロッパーツール] を開いて、[Network] タブを選択した状態で、例えば技術書典 8 のサイト^{*7}を開くと、次のようにたくさんの行が表示（図 5.10）されます。2020 年 2 月時点、このページは 54 往復のリクエストとレスポンスで表示されています。そしてこの 1 行 1 行が「○○をください」「はい、どうぞ」という、リクエストとレスポンスの往復を表しているのです。



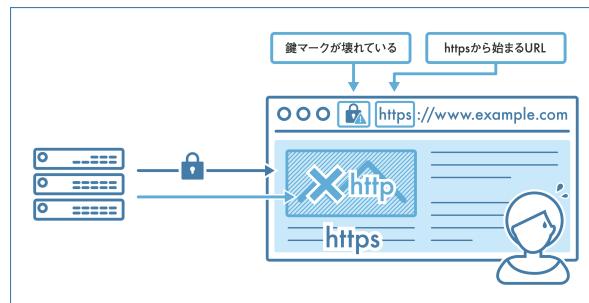
▲図 5.10 54 往復のリクエストとレスポンスで表示された技術書典 8 のページ

5.2.5 HTTP の混在コンテンツはブロックされてしまう

「HTTPS から始まる URL を開いたときに、鍵マークではなく i マークが表示される」という現象には、この「ページの表示は一往復じゃない」という部分が大きく関係しています。

通常、ブラウザで `https://` から始まる URL を開くと、前述のとおり、リクエストしたページは HTTPS で暗号化された状態で届きます。しかし取得したページの HTML で、「``」のように、HTTP の絶対パスで画像を指定する `` タグが含まれていた場合、その画像ファイルは暗号化されていない HTTP で送られてきます。（図 5.11）

*7 <https://techbookfest.org/event/tbf08>



▲図 5.11 ページ内的一部分が HTTP だと鍵マークが壊れてしまう

つまり、せっかくウェブページを HTTPS にしても、そのウェブページの HTML の中で、CSS や画像ファイルの URL を http:// から始まる形式にしていたり、YouTube などのコンテンツを http:// から始まる形式で埋め込んでいると、ページが表示されるまでのたくさんの往復の中で、一部が HTTP になってしまっているので、ブラウザが鍵マークの代わりに i マークを表示して、[このサイトへの接続は完全には保護されていません] と警告（図 5.12）を出すのです。



▲図 5.12 [このサイトへの接続は完全には保護されていません] と警告が出る

この問題は混在コンテンツ（mixed content）と呼ばれています。2020年3月にリリース予定の Chrome のバージョン 81^{*8}からは、この混在コンテンツが存在した場合、対象

^{*8} <https://developers-jp.blogspot.com/2019/11/https.html>, https://security.googleblog.com/2020/02/protecting-users-from-insecure_6.html

のプロトコルが HTTP から HTTPS へ自動的に変更され、さらに HTTPS での読み込みに失敗すると、そのリソース（`http://`で始まる形式）として CSS や画像、埋め込みコンテンツ）がブロックされる変更が予定されています。

混在コンテンツを直すには、例えば``のようになっていた``タグを「``」のようにパスの部分だけにします。これならページを HTTP で開いたときは画像も HTTP で、ページを HTTPS で開いたときは画像も HTTPS で表示されるため、混在コンテンツにはなりません。

あるいは「画像はページとは別のドメイン名なので、パスだけでなくドメイン名から指定しなければいけない」という場合は、``タグを``のようにプロトコルを省略して書くことで、先ほどと同じように、ページを HTTP で開いたときは画像も HTTP で、ページを HTTPS で開いたときは画像も HTTPS で表示されます。このようにして混在コンテンツを解消してやれば、きちんと鍵マークが表示されるようになります。

ちなみに「Google HTML/CSS Style Guide^{*9}」では、このプロトコルを省略する書き方は非推奨とされています。ページが HTTP か HTTPS かに関わらず、画像は HTTPS で表示して構わない、という場合は、``のように指定する方法が推奨されています。

5.2.6 〈トラブル〉SSL 証明書の有効期限がうっかり切れてしまった

2020 年 2 月、マイクロソフトが提供するチャットサービス、「Microsoft Teams」が一時的に使えなくなる障害が発生^{*10}しました。SSL 証明書には、それぞれ 1 年もしくは 2 年の有効期限があります。有効期限が切れる前に更新を行なって、定期的に新しい SSL 証明書へ差し替えなければいけないのですが、更新を怠って有効期限が切れてしまったことでこの障害が発生したようです。

このように SSL 証明書の更新を忘れて、有効期限がうっかり切れてしまうトラブルは昔から後を絶ちません。例えばサーバ監視サービスの Mackerel では、URL 外形監視で、SSL 証明書の残り日数が閾値を下回るとアラートが発報^{*11}されます。カレンダーに更新予定を入れて管理するのもひとつの対策ですが、こうした方法で SSL 証明書の有効期限を監視する方法がより確実です。

^{*9} <https://google.github.io/styleguide/htmlcssguide.html#Protocol>

^{*10} <https://twitter.com/MSFT365Status/status/1224351597624537088>

^{*11} URL 外形監視にて SSL 証明書の有効期限を監視できるようになりました - Mackerel ブログ <https://mackerel.io/ja/blog/entry/2016/01/29/115632>

5.3 SSL 証明書はどうしてあんなに値段に差があるの？

SSL 証明書の仕組みが分かったところで、SSL 証明書のあるあるな疑問、「どうして SSL 証明書はあんなに値段に差があるの？」について少しお話ししておきましょう。

SSL 証明書で検索してみると色々な種類の証明書が出てきますが、値段を比較してみたところ、DigiCert&Symantec は 219,000 円、GMO グローバルサインは 59,800 円、そして本書で買った FujiSSL は 1,000 円でした。^{*12}

なぜこんなに価格差があるのでしょうか。DigiCert&Symantec は知名度が金額に反映されているので高いのでしょうか？ FujiSSL はあまりにも格安ですが、同じ「SSL 証明書」という名前でも中身が何か違うのでしょうか？

5.3.1 同じ「SSL 証明書」でも 3 つの種類がある

実は同じ「SSL 証明書」という名前で呼ばれていても、その実態は 3 種類に分かれています。分かりやすくいうと「高い」「普通」「安い」の 3 種類で、それぞれ「EV 証明書」「OV 証明書」「DV 証明書」という名前で呼ばれています。

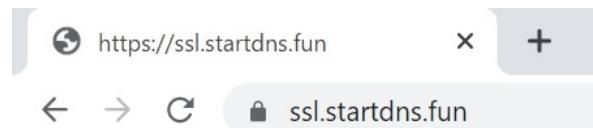
- DV 証明書 (Domain Validation)
 - サイトの運営者がそのドメイン名の持ち主であることを証明してくれる。個人でも取得できる。
 - 当該ドメイン名でメールを受信して URL を踏んだり、TXT レコードを設定したり、サイトに指定のファイルをアップすることで認証される
 - 商品例：FujiSSL、RapidSSL、Let's Encrypt（無料）
- OV 証明書 (Organization Validation)
 - サイトを運営する組織が実在することと、本物の組織であることを証明してくれる。個人では取得できない。
 - サイト運営者の実在をメールと電話で確認することで認証される
 - 商品例：企業認証 SSL (GMO グローバルサイン)、セキュア・サーバ ID (DigiCert&Symantec)
- EV 証明書 (Extended Validation)
 - サイトを運営する法人組織が実在することと、本物の組織であることを証明してくれる。個人や法人格のない組織では取得できない。

^{*12} いずれも 2020 年 2 月時点、有効期間 1 年の税抜金額です

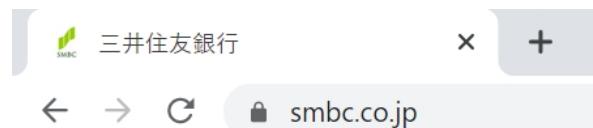
5.3 SSL 証明書はどうしてあんなに値段に差があるの？

- サイトを運営する法人組織の実在を、より厳格に書類と電話で確認することで認証される
- 商品例：SureServer EV（サイバートラスト）、セキュア・サーバ ID EV（DigiCert&Symantec）

このように値段も取得の煩雑さも EV 証明書の方が圧倒的に上なのですが、実際に DV 証明書（図 5.13）と EV 証明書（図 5.14）のサイトを比較すると、残念ながらアドレスバーを見ただけでは何の差もありません。



▲図 5.13 本書で作ったサイトの DV 証明書の鍵マーク



▲図 5.14 三井住友銀行の EV 証明書の鍵マーク

5.3.2 さよならグリーンバー

実は、以前は EV 証明書であれば、どのブラウザでも次のようにグリーンバーでサイトの運営法人が表示（図 5.15）され、一目で本物のサイトと判別ができました。



▲図 5.15 以前はどのブラウザでもグリーンバーが表示されていた

ですが、2019 年 9 月にリリースされた Chrome バージョン 77、そして同年 10 月にリリースされた Firefox バージョン 70 からは、EV 証明書であってもグリーンバーが表示さ

れなくなりました。2020年2月現在、誰でも安価に取得できるDV証明書と、厳格に実在確認をするEV証明書のどちらも、URLの左側にでる鍵マークは同一で、ぱっと見では区別がつかなくなっています。さらに鍵マークをクリックすると、どちらも緑色で【この接続は保護されています】と表示されます。EV証明書の唯一の違いとしては、運営組織名が表示（図5.17）されますが、これをEV証明書の恩恵と見なして高い金額を払うのはつらいだろうな、というのが筆者の印象です。



▲図5.16 DV証明書でも緑色で【この接続は保護されています】と表示される



▲図5.17 EV証明書だと運営組織名も表示される

5.3.3 任意のサブドメインで使えるワイルドカード証明書

SSL 証明書には、「*.example.com」のように、任意のサブドメインで使えるワイルドカード証明書という種類があります。本番環境、ステージング環境、テスト環境などで1つの証明書を共用できるので便利です。

このとき SSL 証明書の SAN が「*.example.com」のみだった場合、元となるドメインや、サブドメインのサブドメインには使用できないので注意が必要です。

- SAN が「*.example.com」のワイルドカード証明書が使えるドメイン名
 - www.example.com
 - prod.example.com
 - stg.example.com
 - test.example.com
- SAN が「*.example.com」のワイルドカード証明書が使えないドメイン名
 - old.www.example.com
 - example.com

認証局によっては「*.example.com」や「www.example.com」の SSL 証明書を買うと、SAN に「example.com」を無料サービス^{*13}で追加してくれるところもあります。取得前によく確認しましょう。なお EV 証明書のワイルドカード証明書は存在しません。

5.3.4 【ドリル】リダイレクトするだけでも www なしの証明書は必要？

問題

次のようなリダイレクトの設定を行なったとします。

1. `http://example.com/` へのアクセスは 4 にリダイレクト
2. `http://www.example.com/` へのアクセスは 4 にリダイレクト
3. `https://example.com/` へのアクセスは 4 にリダイレクト
4. `https://www.example.com/` でサイトを表示

このとき用意すべきなのは、どのドメイン名の SSL 証明書でしょうか？

- A. `www.example.com` の SSL 証明書
- B. `example.com` の SSL 証明書

^{*13} 例えば「FujiSSL ワイルドカード」は「*.example.com」で申し込みと、SAN に「example.com」を自動付加してくれます

- C. example.com と www.example.com の SSL 証明書

答え _____

解答

正解は C です。「リダイレクトする」というレスポンスを返す処理よりも前に、認証や鍵交換が行なわれますので、3 のリダイレクトのために example.com の SSL 証明書も必要です。

5.3.5 <トラブル> サイトを HTTPS 化したら古い端末で別サイトが表示された

1 台のウェブサーバで複数のサイトが同居している環境で、名前ベースのバーチャルホストを HTTPS でも有効にして、すべてのサイトを HTTPS 化しました。パソコンのブラウザで確認したときは、どのサイトも問題なく HTTPS で表示されたのですが、2011 年ごろに買った Android 端末でサイトを見ようとしたところ、なんと同居している別のサイトが表示されてしまいました。

HTTPS の場合、アクセスしたいサイトのドメイン名すら暗号化された状態でサーバにリクエストが届くため、クライアントがアクセスしたいドメイン名をサーバへ伝えるためには、TLS の SNI (Server Name Indication) 拡張機能が必要となります。iOS3 以前の Safari や、Android2.3、WindowsXP 上の IE など、クライアントの端末が著しく古く、SNI に対応していない場合は、アクセスしたいドメイン名がサーバに伝わらず、サーバ側のデフォルトのホストが応答してしまうのです。

対策としてはサーバに IP アドレスを追加して、サイトごとに別々の IP アドレスを割り振ってやるか、もしくは SNI 非対応端末をサイトのサポート対象から外す^{*14}か、のどちらかです。

^{*14} Apache2.4 の場合は、SSLStrictSNIVHostCheck ディレクティブをオンにしておくと、SNI 非対応端末がアクセスしてきたときに、デフォルトホストを出すのではなく、接続自体を拒否して 403 Forbidden になります。 https://httpd.apache.org/docs/current/mod/mod_ssl.html#sslstrictsnivhostcheck

5.4 もっと SSL/TLS を学びたいときは

「はじめに」に書いたとおり、新しい技術を学ぶとき、私たちの理解度は「分かった！」「分かってなかった…」という上下をくり返しながら、ゆるやかに上がっていくものです^{*15}。みなさんも本書を通して SSL/TLS を学び、最初の「分かった」まで辿りつけましたでしょうか？

最初の一歩は、小さな一歩です。でもとても素晴らしい一歩です。本書は、初心者が挫折せずに最初の「分かった」へ辿りつけるところを目指したので、省略した部分もかなり多く、説明は概略の範囲にとどまっています。本書を読み終えて、もっと SSL/TLS について知りたくなったら、次の本をお勧めします。

- プロフェッショナル SSL/TLS Ivan Ristić、齋藤孝道（監訳）
 - <https://www.lambdanote.com/products/tls>
- 暗号技術入門 第3版 結城浩
 - <https://www.hyuki.com/cr/>
- マスタリング TCP/IP SSL/TLS 編 Eric Rescorla、齋藤孝道、古森貞、鬼頭利之
 - <https://www.amazon.co.jp/dp/4274065421>

一回も空振りをせずプロになった野球選手はいません。間違えることを恐れて学ばないより、これからも「分かった！」と「分かってなかった…」を繰り返して、しっぽいねこ^{*16}を抱きしめながら一步ずつ進んでいきましょう。

^{*15} 筆者も本書を書きながら、ずっと「SSL/TLS わかったー！」「私は SSL/TLS について多大な勘違いをしているのでは…？」「いや、SSL/TLS わかってきたかも」「もうだめだ、SSL/TLS のことは何一つわからない」「わか、わかった？ わかっ…あ、いや、わかっ…？」という上下をくり返しました

^{*16} 「技術をつたえるテクニック」の「【コラム】失敗すると「しっぽいねこ」が生まれる」より <https://booth.pm/ja/items/1316755>

あとがき

好きな技術の本を書くのは、とても楽しいものです。「誰に頼まれた訳でもないのになぜ書くの？」と自分に問いかけると、そこには、「これを書いたら、きっと誰かの助けになるはずだ」という祈りのような希望があります。

0から1を作るのはとても大変で、挫けそうになるときもあります。以下は、挫けそうになっていた筆者に対して、同僚であり友人でもある Marshall が送ってくれたメッセージです。書くことで誰かの役に立とうとしている人の胸に、そっと灯りをともすようなメッセージだったので、あとがきに載せる許可をもらいました。

快く承諾してくれた Marshall に感謝します。

To be honest, people can find **ANY** information on the internet. Everything I learned at my university I could have learned from the internet--I still went to university.

There is a lot of information floating around online. There is a lot of information about marketing, but I still have a ton of books at home on how to become a better marketer.

People can take pictures with their smartphone--that doesn't mean people who draw should stop drawing or people who paint should stop painting.

Keep writing!

I've written a lot of articles online with the intention of helping people. People will leave comments like "What a stupid article! You think people don't already know how to do this? Why waste your time writing this?!"

But my answer is because I know my article helped someone and that makes me happy. Not everyone knows everything and if I can share my knowledge and help at least one person, then it is worth it for me.

The point I'm trying to make is no matter what you do there is someone who will try to make you feel bad about it. Keep doing it anyway.

数ある技術書の中から「SSLをはじめよう」を手に取ってくださったあなたに感謝します。

2020年3月
mochikoAsTech

PDF版のダウンロード

本書（紙の書籍）をお買い上げいただいた方は、下記のURLからPDF版を無料でダウンロードできます。

- ダウンロード URL
 - <https://mochikoastech.booth.pm/items/1834443>
- パスワード
 - xxxxxxxx

Special Thanks:

- ふさふさ茶色のやさしい母猫に捧ぐ
- いつも変わらぬ愛で包んでくれる旦那様
- なんでもない日を祝ってくれる息子
- Gunnell Marshall

レビュアー

- Takeshi Matsuba
- Mari Kubota
- およねさん
- 矢崎 誠

参考文献・ウェブサイト

- プロフェッショナルSSL/TLS Ivan Ristić、齋藤孝道（監訳）
 - <https://www.lambdanote.com/products/tls>
- プロフェッショナルIPv6 小川晃通

-
- <https://www.lambdanote.com/products/ipv6>
 - 食べる！ SSL！ —HTTPS 環境構築から始める SSL 入門 小島拓也、中嶋亜美、吉原恵美、中塚淳
 - <https://www.amazon.co.jp/dp/B00PHC4480>
 - SSL/TLS の基本 - Qiita
 - https://qiita.com/angel_p_57/items/446130934b425d90f89d
 - 【図解】初心者も分かる"公開鍵/秘密鍵"の仕組み～公開鍵暗号方式の身近で具体的な利用例やメリット～ | SE の道標
 - <https://milestone-of-se.nesuke.com/sv-advanced/digicert/public-private-key/>

著者紹介

mochiko / @mochikoAsTech

元 Web 制作会社のインフラエンジニア。技術書典で出した本がきっかけで、テクニカルライターの仕事を始めた。「分からない気持ち」に寄り添える技術者になれるように日々奮闘中。技術書典 4~7 で頒布した「DNS をはじめよう」「AWS をはじめよう」「技術をつたえるテクニック」「技術同人誌を書いたあなたへ」は累計で 9,700 冊を突破。

- <https://twitter.com/mochikoAsTech>
- <https://mochikoastech.booth.pm/>
- <https://note.mu/mochikoastech>
- <https://mochikoastech.hatenablog.com/>

Hikaru Wakamatsu

表紙デザインを担当。

Shinya Nagashio

挿絵デザインを担当。

SSLをはじめよう

「なんとなく」から「ちゃんとわかる！」へ

2020年3月1日 技術書典8 初版

著 者 mochikoAsTech

デザイン Hikaru Wakamatsu / Shinya Nagashio

発行所 mochikoAsTech

印刷所 日光企画

(C) 2020 mochikoAsTech