

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

本著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？ 本著を書くまで、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。たとえば「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「HTTP は普通、HTTPS にすれば安心」から、「HTTP は危険、HTTPS が普通」に変わってきました。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あ…」と思っている。本著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。まずは本著で、一緒に最初の「分かった！」まで進んでみましょう。

ちなみに本著「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、本著では最低限にとどめています。本著で 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。

DNS 本、AWS 本、そして SSL 本のはじめよう 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

本著は、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

本著は、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

本著の特徴

本著では実際にサーバを立てて SSL 証明書の設置を行い、HTTPS のサイトを作ってみます。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。

本著のゴール

本著を読み終わると、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

本著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。

また本著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。よって本著の記載内容に基づいて読者が行った行為、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	4
本著の特徴	4
本著のゴール	5
免責事項	5
第1章 Oracle Cloud のアカウントを作ろう	11
1.1 サーバを立てる下準備	12
1.1.1 サイトを作るのにどうしてサーバがいるの？	12
1.1.2 サーバを立てるにはお金が必要？	13
1.1.3 なんで AWS じゃなくて Oracle のクラウドを使うの？	13
1.2 Oracle Cloud でアカウント登録	14
1.2.1 無料でアカウントを作ろう	14
1.2.2 〈トラブル〉 どうしても SMS が届かない！ そんなときは？	21
1.2.3 パスワードと支払情報の登録	21
1.2.4 Oracle Cloud のコンソールにサインイン	26
第2章 Oracle Cloud でサーバを立てよう	29
2.1 事前準備	30
2.1.1 Windows で RLogin をインストールする	30
2.1.2 Windows で SSH のキーペア（秘密鍵・公開鍵）を作成する	32
【コラム】 SSH の秘密鍵にパスフレーズは設定すべき？	37
2.1.3 Mac でターミナルを準備する	38
2.1.4 Mac で SSH のキーペア（秘密鍵・公開鍵）を作成する	40
【コラム】 ターミナルでコピー&ペーストするには？	41

2.2	コンピュートでサーバを立てる	41
2.2.1	OS は Oracle Linux 7.7 を使おう	43
2.2.2	作っておいた SSH の公開鍵を設置しよう	43
2.2.3	〈トラブル〉 "Out of host capacity." が起きたらどうすればいい？	44
2.2.4	Always Free ではなく無償クレジットの枠でサーバを立てよう	45
2.2.5	サーバが起動するまで待とう	48
	【コラム】 Oracle Cloud のコンピュートの金額計算方法	49
	【コラム】 Oracle Cloud と AWS はどっちが安い？	49
2.2.6	接続先となるサーバの IP アドレス	50
第 3 章	ウェブサーバの設定をしよう	53
3.1	サーバに SSH でログインしよう	54
3.1.1	Windows の RLogin を使ってサーバに入ってみよう	54
3.1.2	Mac のターミナルを使ってサーバに入ってみよう	61
3.2	ターミナルでサーバを操作・設定してみよう	62
3.2.1	プロンプトとは？	62
	【コラム】 ターミナルを閉じたいときは？	64
3.3	NGINX をインストールしよう	64
3.4	Firewalld で HTTP と HTTPS を許可しよう	66
3.5	OS を再起動してみよう	67
3.5.1	ターミナルはなんのためにある？	68
3.6	なぜか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 証明書を取得しよう	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	78
	【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	80
	【コラム】 openssl コマンドのユーティリティ（道具）たち	82
4.3.1	【ドリル】 CSR で入力すべきなのはクライアントの情報？	83
4.4	SSL 証明書の取得申請を出そう	84
4.5	DNS の設定をしよう	92

4.6	SSL 証明書をサーバに設置しよう	96
4.6.1	Windows で証明書と中間 CA 証明書をアップしよう	96
4.6.2	Mac で証明書と中間 CA 証明書をアップしよう	98
4.6.3	証明書を 1 ファイル (startssl.crt) にまとめよう	99
4.7	NGINX で HTTPS のバーチャルホストを作ろう	100
4.8	HTTPS でサイトを開いてみよう	103
	【コラム】ロードバランサでも SSL ターミネーションできる	103
第 5 章	SSL/TLS について学ぼう	105
5.1	SSL/TLS とは?	106
5.1.1	SSL と TLS はどういう関係?	106
5.1.2	SSL イコール HTTPS ではない	106
5.2	「サイトを HTTPS 化する」とは何か?	107
5.3	どんなサイトでも必ず HTTPS にしなきゃだめ?	107
5.4	HTTP のままだと起きるデメリット	108
5.4.1	サイトが「安全でない」と表示されてしまう	108
5.4.2	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 .	109
5.4.3	相対的に検索順位が下がる	109
5.4.4	周りが HTTPS になるとリファラが取れなくなる	110
5.5	HTTPS 化すると得られるメリット	110
5.5.1	HTTP/2 との組み合わせで速度が向上するケースもある	110
5.5.2	SameSite の変更に対応できる	112
5.5.3	重要情報のアタリが付けにくくなる	112
5.6	SSL 証明書とは	112
5.6.1	SSL サーバ証明書と SSL クライアント証明書	113
5.6.2	SSL 証明書はどんな場面で使われている?	113
5.7	SSL 証明書は異なる 3 つの仕事をしている	114
5.7.1	HTTPS の実際の流れ	115
5.7.2	ウェブページは 1 往復で表示されるわけじゃない	117
5.7.3	HTTP の混在コンテンツはブロックされてしまう	118
5.7.4	正当性を証明する中間 CA 証明書	120
5.7.5	ルート証明書はトラストストアにある	122
5.7.6	〈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 .	124
5.8	SSL 証明書はどうしてみんなに値段に差があるの?	124
5.8.1	同じ「SSL 証明書」でも 3 つの種類がある	124

目次

5.8.2 さよならグリーンバー	126
5.8.3 任意のサブドメインで使えるワイルドカード証明書	127
5.8.4 【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？	128
5.8.5 <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	128
あとがき	131
PDF 版のダウンロード	132
Special Thanks:	132
レビュー	132
参考文献・ウェブサイト	132
著者紹介	135

第 1 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。

実際に SSL 証明書を取得して、HTTPS のサイトを作る、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。

1.1 サーバを立てる下準備

HTTPSでサイトを作るのに必要な材料は次の3つです。

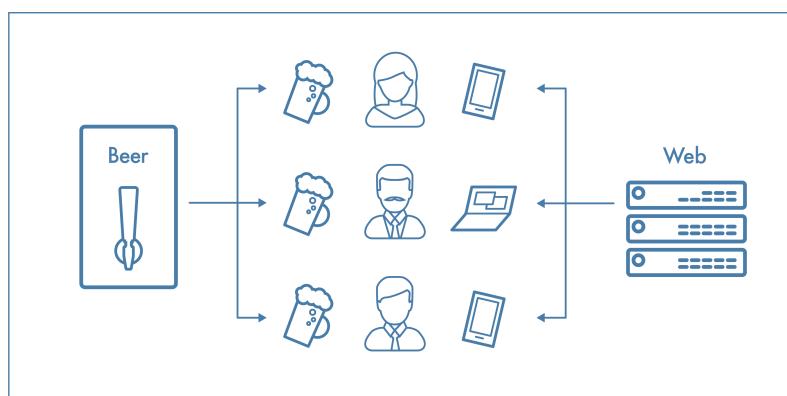
- ウェブサーバ
- ドメイン名
- SSL証明書

まずは1つめのウェブサーバを立てるため、下準備からはじめます。

1.1.1 サイトを作るのにどうしてサーバがいるの？

ところで、これからウェブサーバを立てようとしていますが…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…つまりコック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1.1）



▲図1.1 ビアサーバもウェブサーバもリクエストしたらサービスが提供される

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、それを載せておく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でしか見られず、イン

インターネットで公開できないのです。^{*1}

というわけでウェブサイトを提供するために、まずはウェブサーバを立てましょう！

1.1.2 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。ですがオラクルがやっている「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、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。本著ではこの 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やっているクラウドですが、そもそもクラウドがなんだか分からぬといまいちピンと来ないかもしれません。あなたが「ウェブサイト公開したいなあ…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っ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、ブラウザでぽちぽちとスペックを選んでいくだけで、すぐにサーバが使えます。

1.1.3 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。

^{*1} サーバについては、はじめようシリーズの 2 冊目、「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

^{*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.com>

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。

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。

たくさんのクラウドがある中でどこを選ぶのか、その理由は、本来であればその上で動かすサービス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？

本著では次の 2 つを目的として、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

1.2 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）

なお本著では、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。

1.2.1 無料でアカウントを作ろう

「Oracle Cloud 無料」で検索（図 1.2）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} jp/blog/19/iaaspaaawsawsazuregooglealibaba4720193synergy_research_group.html

^{*9} 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/>

1.2 Oracle Cloud でアカウント登録



▲図 1.2 「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 1.3）

^{*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 つを内包した総称が「Free Tier（無償ティア）」のようです

第1章 Oracle Cloud のアカウントを作ろう



▲図 1.3 [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/地域] を入力して、使用条件を確認した上で [次] をクリックしましょう。(図 1.4) 後で分からなくなないように、登録した項目を表 1.1 にメモしておきましょう。

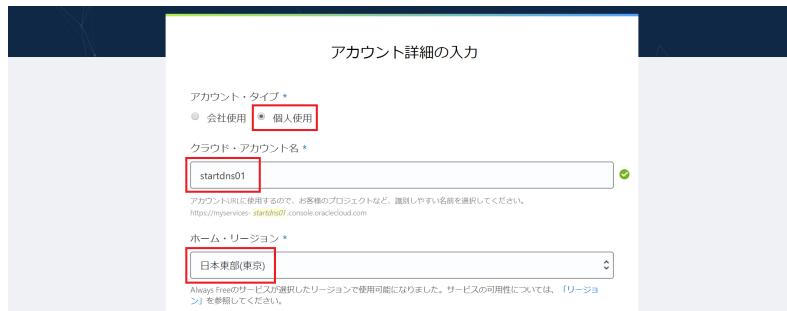
▼表 1.1 Oracle Cloud に登録した情報

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 1.4 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。(表 1.2) 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。この【クラウド・アカウント名】は、後で管理画面にサインインするときのアカウント URL になります。(図 1.5)



▲図 1.5 【クラウド・アカウント名】には好きな名前を入力

【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、**各地域の中でどこを使うか？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っ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。ただし Oracle Cloud のサービスによってはまだ東京リージョンで使えないものもあります。その場合は次点として「米国東部(アッシュバーン)」を選択してください。

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 のサービスは、選択した【ホーム・リージョン】においてのみ利用できます。

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【米国東部(アッシュバーン)】を選択してください。本著では Always Free が使えなかった場合は、300ドル分の無償クレジットを使用しますので、【日本東部(東京)】を選択して問題ありません。

▼表 1.2 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)	

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】をすべて入力できましたか？（図 1.6）

The screenshot shows a registration form with various input fields. The following fields have red boxes drawn around them:

- 姓 *: 'もち子' (Mochiyo)
- 名 *: '猫村' (Nekomura)
- 部門名 *: '個人' (Individual)
- 住所 *: '新宿四丁目1番6号' (Shinjuku-ku, 4-chome, 1-6) and 'JR新宿ミライナタワー' (JR Shinjuku Miraina Tower)
- 市区町村 *: '新宿区' (Shinjuku-ku)
- 郵便番号 *: '160-0022'
- 都道府県 *: 'TOKYO'
- 国/地域: '日本' (Japan)

▲図 1.6 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】をクリックしてください。(図 1.7)

The screenshot shows a form for mobile phone number verification. The following fields are visible:

- モバイル番号 *: A dropdown menu showing '日本 (81)' and a text input field containing '90' followed by a blacked-out area.
- A note below the dropdown states: 'ご契約トロリーフにて機回し必要な場合、オラクル社はお客様に機回しを禁じることがあるため、リスト削除のモバイル番号が使用できません。この機回しコードは、テキスト・メッセージで携帯電話に送信されます。標準テキスト・メッセージレートが適用されます。' (If you are using a service provider that restricts roaming, Oracle will prohibit it. This roaming code will be sent via text message to your mobile phone. Standard text message rates apply.)
- A note at the bottom left: '上記のモバイル番号および電子メール startdn10@gmail.com に基づく検証コードがすでにある場合、[ここをクリックしてコードを確認してください。](#)' (If a verification code has already been issued for the above mobile number and email address, click [here](#) to check it.)
- A large blue button at the bottom: '次: モバイル番号の確認' (Next: Mobile number confirmation).
- At the very bottom: '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](#)' (Do you need support? Please contact us: [Chat Support](#)).

▲図 1.7 携帯電話の番号を入力

数分以内に [Your Oracle Cloud verification code is ○○○○○○○○.] と書かれた SMS が届きます。(図 1.8)



▲図 1.8 コードの書かれた SMS が届いた

SMS で届いた「〇〇〇〇〇〇〇〇」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 1.9)



▲図 1.9 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

1.2.2 〈トラブル〉 どうしても SMS が届かない！ そんなときは？

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。たと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}

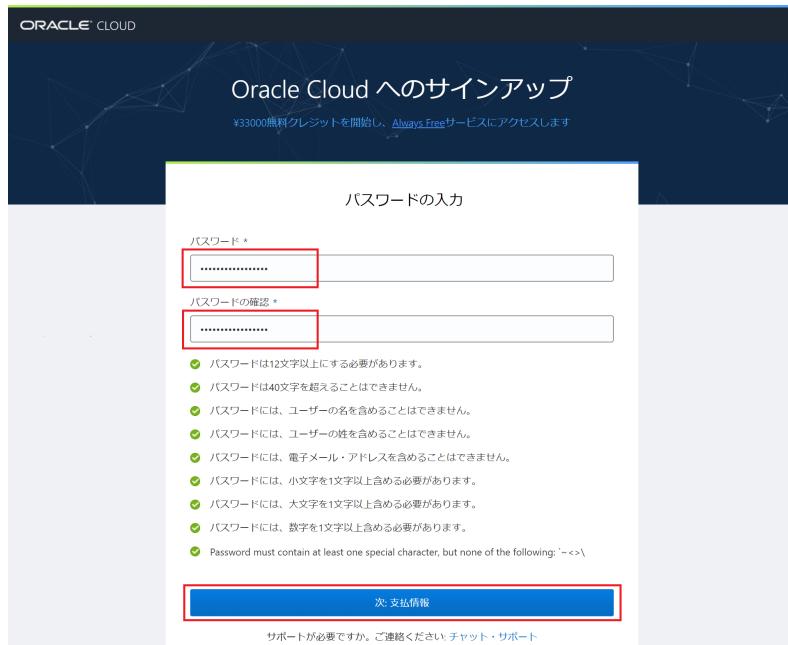
ちなみに筆者の場合は、特に設定変更をせず同じ番号で 2 回試してみたのですが、最初は届かず、もう 1 回試してようやく届きました。

迷惑メールの設定を確認して何回か試して、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。

1.2.3 パスワードと支払情報の登録

正しいコードが確認できると、[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 1.10)

^{*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と思ってください



▲図 1.10 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。(図 1.11) 繰り返しあ伝えしているとおり、Oracle Cloud には Always Free という無料枠があります。本著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つてから 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



▲図 1.11 [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 1.12)*¹³

The form is titled 'カード情報'. It has fields for 'カードの種類' (Visa, Mastercard, Amex, JCB) with Visa selected. There are two input fields for 'カードの番号' and '有効期限'. A note below says: '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。' At the bottom right is a green 'Finish' button.

▲図 1.12 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 1.13)

*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

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 1.13 チェックを入れて [サインアップの完了] をクリック

これでアカウント登録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 1.14）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>

1.2 Oracle Cloud でアカウント登録

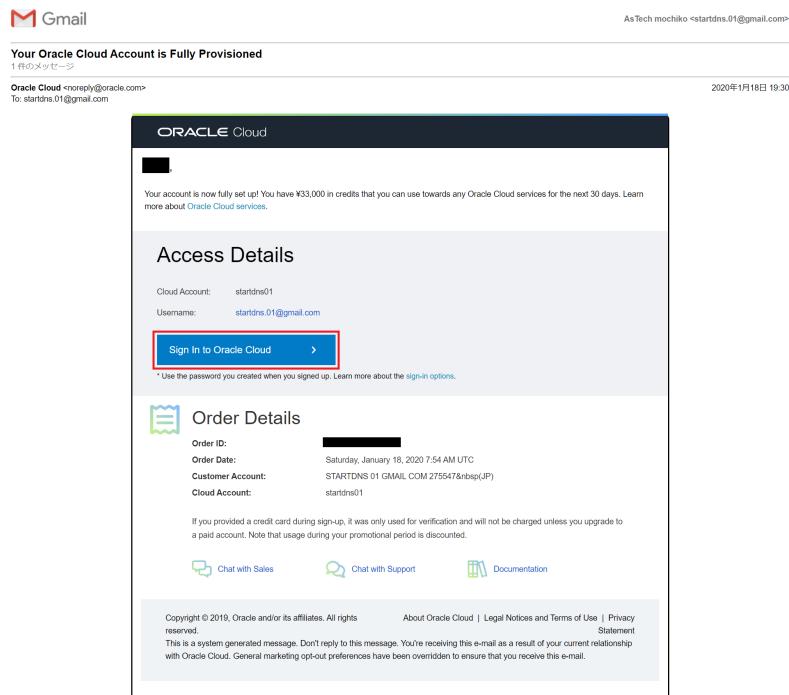


▲図 1.14 アカウント登録の手続きはおしまい

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。メールの [Sign In to Oracle Cloud] をクリックしましょう。(図 1.15)

^{*15} 筆者の場合は、メールが届くまで 2 時間半かかりました

第1章 Oracle Cloud のアカウントを作ろう



▲図 1.15 数時間後、準備完了を知らせるメールが届く

1.2.4 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 1.16) [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

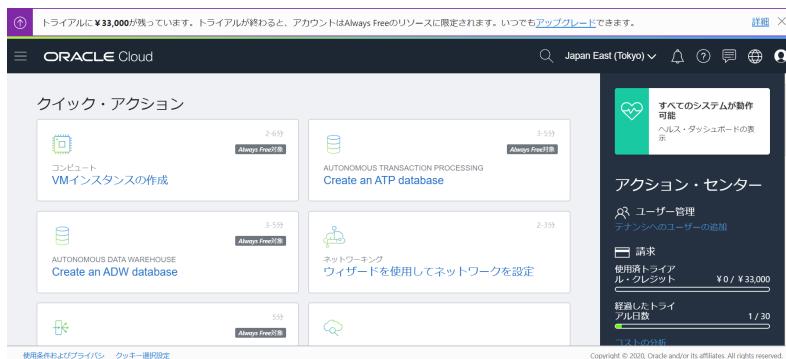
*17 メールにも書いてありますが、ここでの [ユーザー名] とは [クラウド・アカウント名] (筆者の場合は startdns01) ではなく、[メールアドレス] のことです。紛らわしいのでご注意ください

1.2 Oracle Cloud でアカウント登録



▲図 1.16 [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！ コンソールにサインインできました。



▲図 1.17 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、右上の人マークから「クラウドにサインイン」をクリックしましょう。

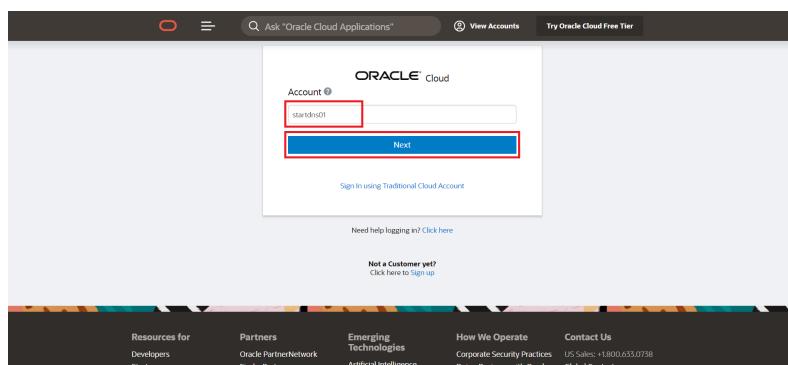
*18 <https://www.oracle.com/jp/>

第1章 Oracle Cloud のアカウントを作ろう



▲図 1.18 右上的人物マークから [クラウドにサインイン] をクリック

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 1.19 [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での、数ページ戻って確認してみましょう

第 2 章

Oracle Cloud でサーバを立てよう

HTTPS で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

この章では、Oracle Cloud でウェブサーバを立てていきます。

2.1 事前準備

2.1.1 Windows で RLogin をインストールする

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。サーバに接続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、本著ではいちばん上の RLogin (図 2.1) を使って説明していきます。

- RLogin (<http://nanno.dip.jp/softlib/man/rlogin/>)



▲図 2.1 RLogin

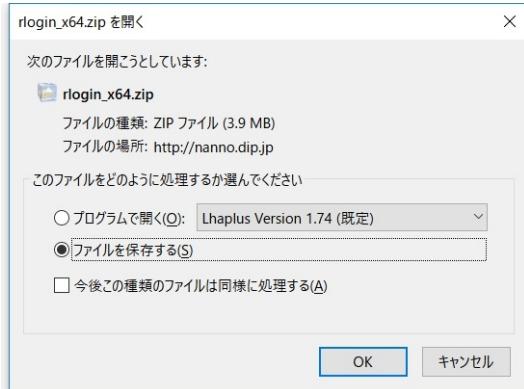
RLogin の「実行プログラム (64bit)*1」(図 2.2) の URL、http://nanno.dip.jp/softlib/program/rlogin_x64.zip をクリックしてください。



▲図 2.2 「実行プログラム (64bit)」の URL をクリックしてダウンロード

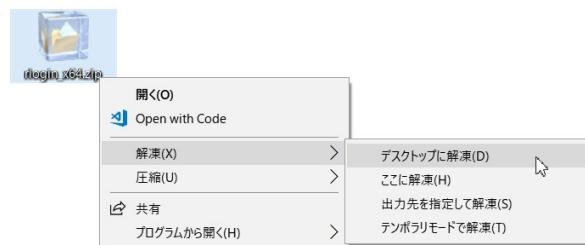
*1 もしパソコンの Windows が 32bit 版だった場合は「実行プログラム (32bit)」の URL をクリックしてください。

ダウンロードした ZIP ファイルを保存（図 2.3）します。保存場所はどこでも構いませんが、後でどこに置いたか分からなくなりそうな人はデスクトップに保存しておきましょう。



▲図 2.3 「ファイルを保存する」でパソコンに保存

デスクトップの ZIP ファイル（rlogin_x64.zip）を右クリック（図 2.4）して、[解凍]>[デスクトップに解凍]*2をクリックします。



▲図 2.4 ZIP ファイルを右クリックして解凍>デスクトップに解凍

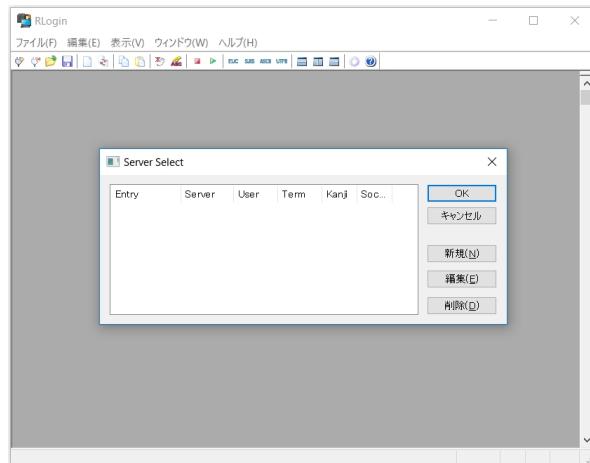
解凍したら、デスクトップにできた「rlogin_x64」というフォルダの中にある「RLogin.exe」*3（図 2.5）をダブルクリックすれば RLogin が起動（図 2.6）します。

*2 ZIP ファイルを右クリックしても「解凍」が見当たらないときは、圧縮・解凍の定番ソフトである Lhaplus をインストールしましょう。 <https://forest.watch.impress.co.jp/library/software/lhaplus/>

*3 フォルダの中に RLogin はあるけど RLogin.exe なんて見当たらない…という場合、ファイルの拡張子が非表示になっています。この後も拡張子を含めてファイル名を確認する場面が何度かでできますので、



▲図 2.5 RLogin.exe をダブルクリック



▲図 2.6 RLogin が起動した

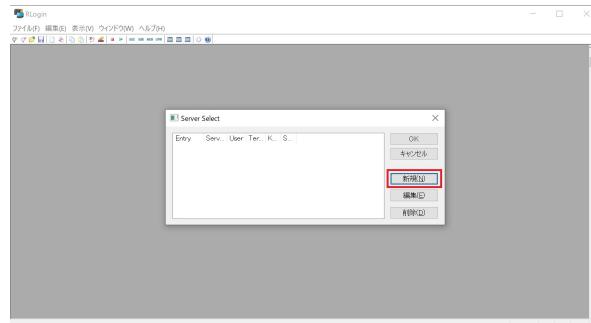
これで RLogin のインストールは完了です。

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

Windows の方は、続いて起動した RLogin で「新規 (N)」をクリックします。

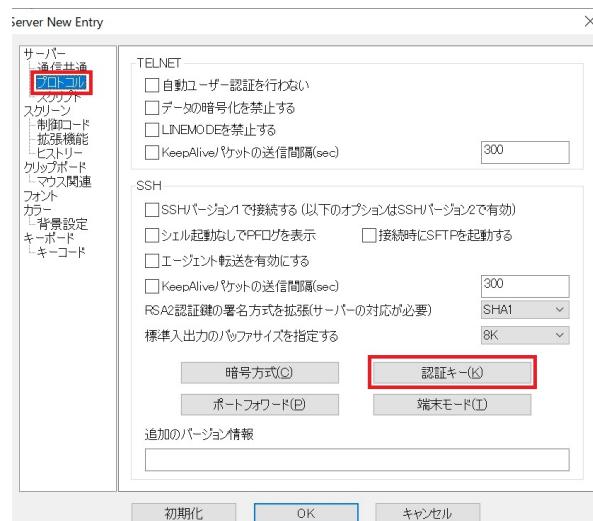
表示されていない人は「拡張子 表示」で Google 検索して、拡張子が表示されるように設定変更しておきましょう。

2.1 事前準備



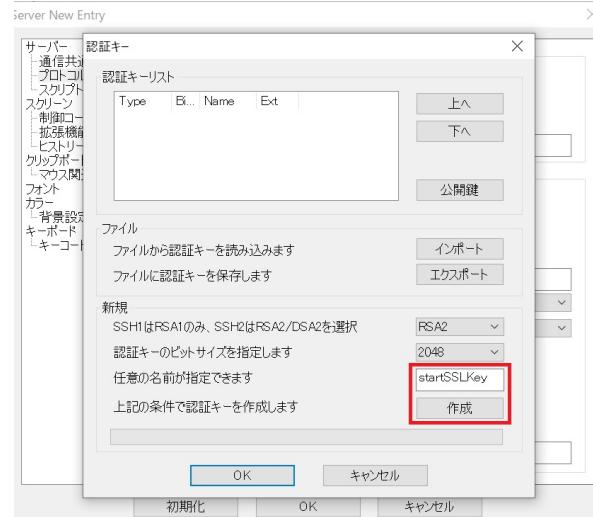
▲図 2.7 [新規 (N)] をクリック

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。



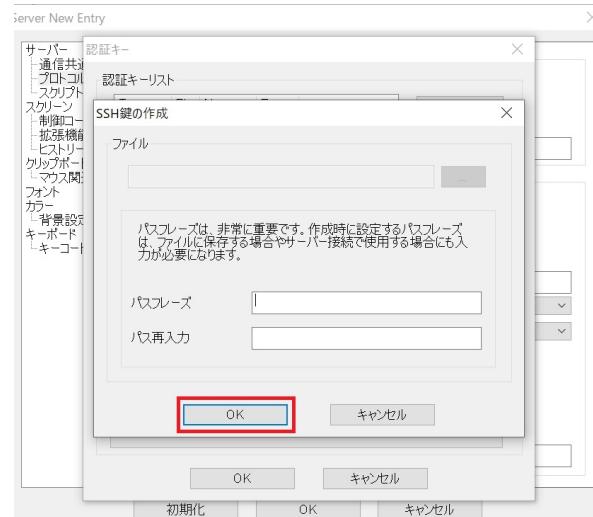
▲図 2.8 [プロトコル] を選択して [認証キー (K)] をクリック

[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 2.9 [startSSLKey] を入力して [作成] をクリック

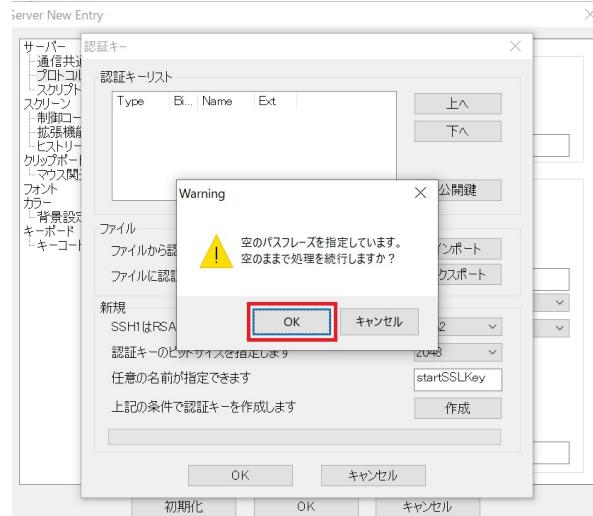
[パスフレーズ] と [パス再入力] には何も入力せず、[OK] をクリックします。^{*4}



▲図 2.10 何も入力せず [OK] をクリック

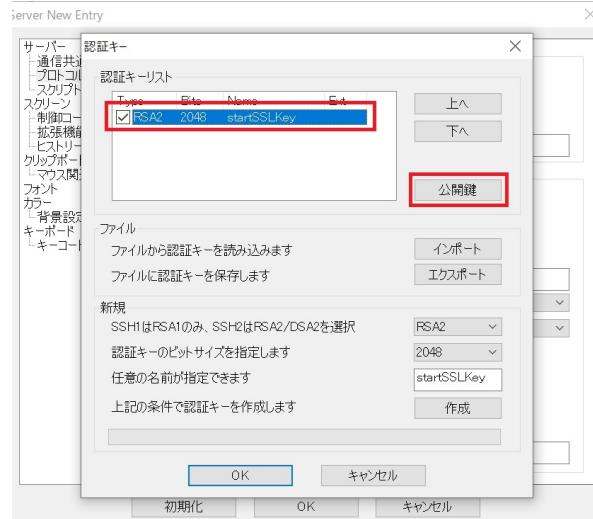
^{*4} このパスフレーズは秘密鍵の保護に用いられます。パスフレーズを設定しなくてよいのか、についてはコラムで後述します

【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。



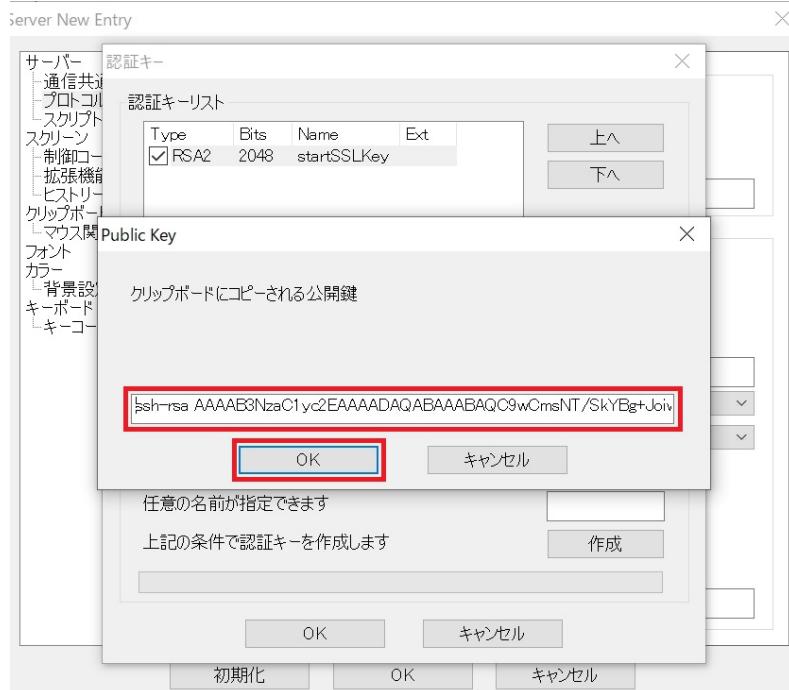
▲図 2.11 [OK] をクリック

【認証キーリスト】に、いま作った [startSSLKey] が表示されたら、キーペア（秘密鍵・公開鍵）が無事できています。【公開鍵】をクリックしてください。（図 2.5）



▲図 2.12 [startSSLKey] が表示されたら [公開鍵] をクリック

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 2.13 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 はいったん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。メモした公開鍵も無くさないようご注意ください。

【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ってしまっても、暗証番号が分からなければロック画面が解除できず、勝手に使えないので

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要るの…？」という具合に、初心者を混乱に陥れやすいので、本著では秘密鍵をパスフレーズなしで作って使います。

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。

2.1.3 Mac でターミナルを準備する

Mac^{*5}を使っている方は、最初から「ターミナル」(図 2.14) 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。

^{*5} 2020 年 2 月以前にリリースされた macOS を対象とします



▲図 2.14 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 2.15）すれば起動できます。



▲図 2.15 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

2.1.4 Mac で SSH のキーペア（秘密鍵・公開鍵）を作成する

Mac の方は、ターミナルで次のコマンドを実行してください。⁶

```
$ 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⁷で公開鍵を表示してみましょう。

⁶ 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

⁷ cat は猫ではなく「conCATenate files and print on the standard output」の略です

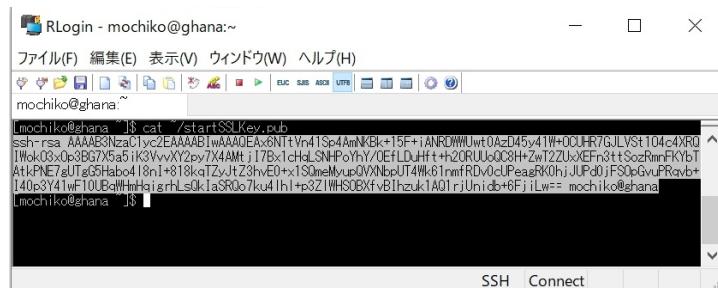
```
$ cat ~/startSSLKey.pub
ssh-rsa AAAAB3NzaC1yc2E=... mochikoAsTech@mochikoMacBook-Air.local
```

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】ターミナルでコピー＆ペーストするには？

ターミナルで表示されている内容をコピーしたいときは、コピーしたい部分を **マウスで選択するだけ**です。（図 2.16）選択してから Ctrl+c を押す必要はありません。



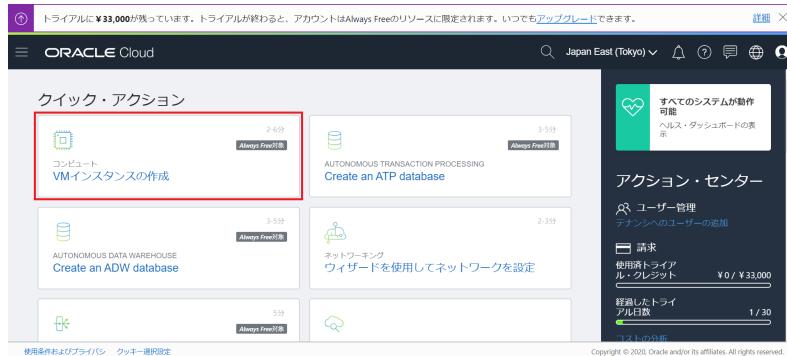
▲図 2.16 マウスで選択するだけでコピーできる

逆にコピーした内容をターミナルへペーストしたいときはターミナル上で**右クリックするだけ**です。ターミナルのソフトにもよりますが、基本的に Ctrl+v は使えないで注意してください。

2.2 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 2.17）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 2.17 [VM インスタンスの作成] をクリック

[インスタンスの命名] に [startSSLInstance] と入力します。(図 2.18) 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、何も変更せずそのまま構いません。



▲図 2.18 [インスタンスの命名] に [startSSLInstance] と入力

2.2.1 OS は Oracle Linux 7.7 を使おう

パソコンには OS 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。

本著では、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 Linuxでサーバを立てるときは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

名前のとおり、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、使い勝手はほぼ同じです。

2.2.2 作っておいた SSH の公開鍵を設置しよう

さらに下に進んで [SSH キーの追加] は、[SSH キーの貼付け] を選択して、そこに先ほどメモしておいた公開鍵をペーストします。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 2.19)



▲図 2.19 [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。

2.2.3 <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つたよ？」という方は、このコラムは読み飛ばして、この先の「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。



▲図 2.20 "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 で立てようとしたサーバは、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 2.21 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、定期的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 の枠でサーバは立てられません。

- 自分以外のユーザーがサーバ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たら、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！

2.2.4 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。

第2章 Oracle Cloud でサーバを立てよう

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)] という種類のサーバでした。"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] をクリックしましょう。(図 2.22)



▲図 2.22 [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、サーバスペックごとに「シェイプ」という区分があります。^{*8} 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] から変更したいので [シェイプの変更] をクリックしてください。

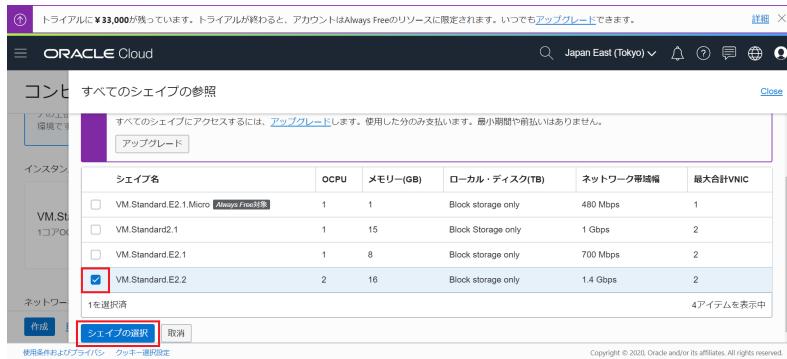


▲図 2.23 [シェイプの変更] をクリック

^{*8} シェイプとはサーバスペックごとの区分のことです。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

2.2 コンピュートでサーバを立てる

OCPU^{*9}が2、メモリが16GBの[VM.Standard.E2.2]^{*10}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 2.24 「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。



▲図 2.25 [作成]をクリック

^{*9} 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 のサーバを使っている。同等スペックのサーバを用意してほしい」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。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

^{*10} 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 IO) 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

2.2.5 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...]」と表示されたら、サーバが用意されるまでそのまま数分待ちましょう。



▲図 2.26 「[プロビジョニング中...]」と表示されたら数分待つ

サーバができあがると、表示が緑色の「[実行中]」に変わります。おめでとうございます！ これでサーバが立てられました！



▲図 2.27 「[実行中]」に変わった！ サーバが立てられた！

【コラム】Oracle Cloud のコンピュートの金額計算方法

ところで、いま立てた「VM.Standard.E2.2」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 を超えてしまわないか、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^{*11}を見てみると、[VM.Standard.E2.2] は [[\$0.03]^{*12}] 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 につき 1 時間あたりかかる金額です。^{*13}

「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。

「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^{*14}なので、日本円だと 5356.8 円ですね。

【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、同じスペックのサーバで AWS と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 2.1)

^{*13} <https://www.oracle.com/jp/cloud/compute/pricing.html>

^{*14} 2020 年 1 月時点の金額

^{*15} ちなみに AWS は、同スペックのサーバ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

^{*16} \$1 を 120 円で換算すると \$44.64 * 120 円で 5356.8 円です

▼表 2.1 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 が最適なのであれば 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筆者はお勧めします。

2.2.6 接続先となるサーバの IP アドレス

無事にサーバが「実行中」になったら、接続先となるサーバの IP アドレスを確認してみましょう。

先ほど作成したインスタンス [startSSLInstance] の、[プライマリ VNIC 情報]（図 2.28）にある [パブリック IP アドレス] をメモ（表 2.2）してください。

The screenshot shows the Oracle Cloud Infrastructure (OCI) console interface. At the top, there's a message about trial credits and account status. Below it, the Oracle Cloud logo and a search bar for 'Japan East (Tokyo)'. The main content area has two sections: 'Instances Information' and 'Primary VNIC Information'. In the 'Primary VNIC Information' section, the 'Public IP Address' field is highlighted with a red box and contains the value '140.238.33.51'. Other visible information includes private IP '10.0.0.2', internal FQDN 'startsslinstance...', and security group 'None'.

▲図 2.28 [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 2.2 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入ってみましょう。

第3章

ウェブサーバの設定をしよう

サーバを立てたらそこに NGINX をインストールします。

ウェブサーバの設定ができたら ドメイン名を紐付けて、HTTP でアクセスしてみましょう。

3.1 サーバに SSH でログインしよう

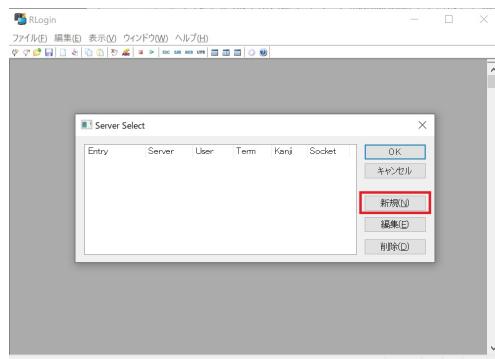
では立てたばかりのサーバに、SSH でログインしてみましょう。

3.1.1 Windows の RLogin を使ってサーバに入ってみよう

Windows のパソコンを使っている方は、デスクトップの [rlogin_x64] というフォルダの中にある [RLogin.exe]（図 3.1）をダブルクリックして RLogin を起動（図 3.2）してください。起動したら [新規] をクリックします。



▲図 3.1 RLogin.exe をダブルクリック

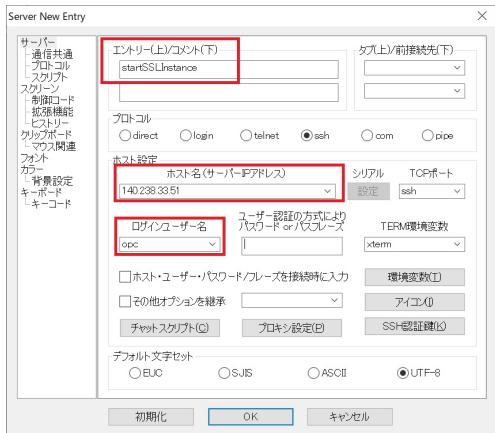


▲図 3.2 RLogin が起動したら [新規] をクリック

初めに [エントリー (上) /コメント (下)] の上に [startSSLInstance] と入力します。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 3.3）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

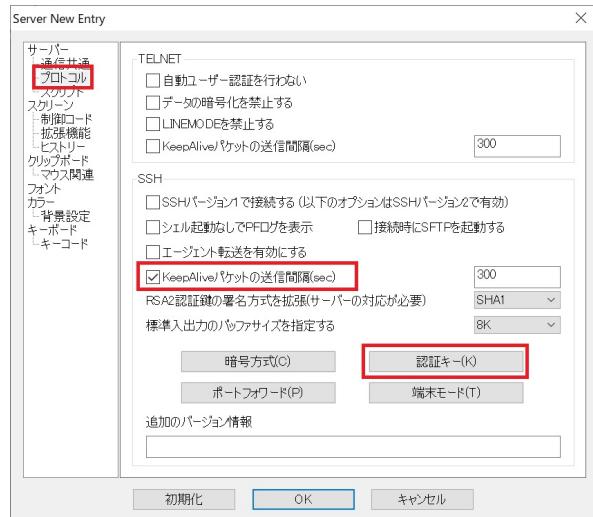
3.1 サーバに SSH でログインしよう

というの 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 3.3 [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

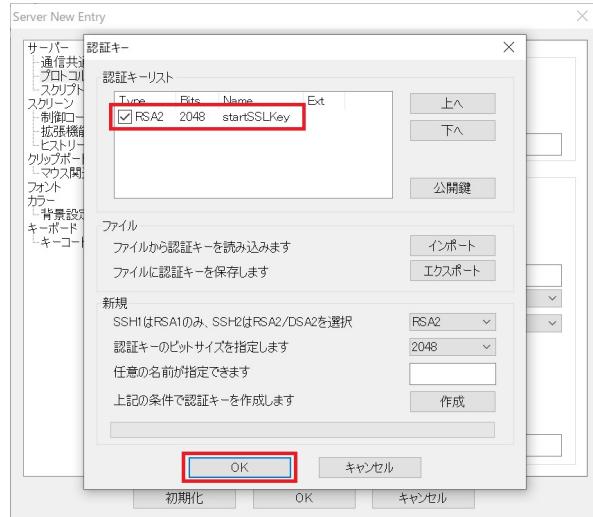
次に左メニューで [プロトコル] を選択（図 3.4）したら、[KeepAlive パケットの送信間隔 (sec)] にチェックを入れておきます。これを設定しておくとターミナルをしばらく放っておいても接続が勝手に切れません。続いて [認証キー] をクリックします。



▲図 3.4 [KeepAlive パケットの送信間隔 (sec)] にチェックを入れて [認証キー] をクリック

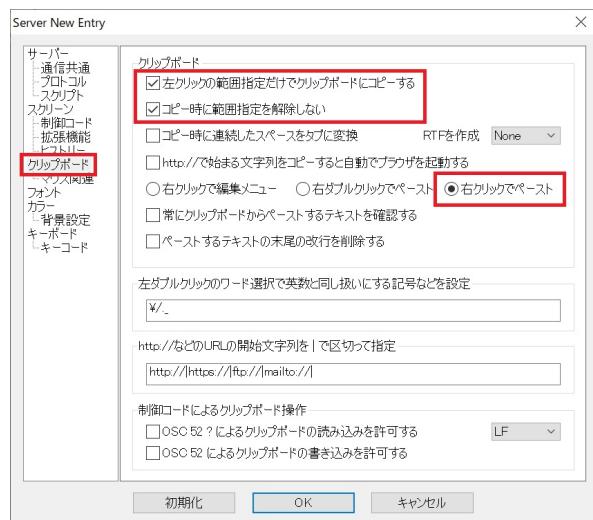
[認証キー] リストで [startSSLKey] にチェックが入っていることを確認（図 3.5）します。これは「ログインするときにこの鍵を使います」というリストです。チェックが入っていたら [OK] をクリックして閉じて構いません。

3.1 サーバに SSH でログインしよう



▲図 3.5 [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 3.6）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 3.6 右クリックや左クリックの設定

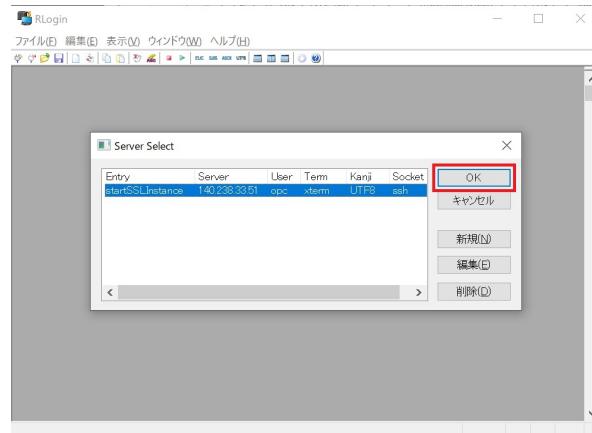
第3章 ウェブサーバの設定をしよう

次に左メニューで「フォント」を選択（図3.7）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3.7 文字セットを「UTF-8」に変更

設定が保存できたら「OK」をクリック（図3.8）してください。



▲図3.8 設定が保存できたら「OK」をクリック

すると初回のみ、この「公開鍵の確認」が表示（図3.9）されます。これは「初めて入

3.1 サーバに SSH でログインしよう

るサーバだけど信頼していいですか？ 本当に接続しますか？」と聞かれているので、「接続する」をクリックしてください。サーバにはそれぞれフィンガープリントという固有の指紋があるため、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れば RLogin が覚えていてくれて、次回以降は「これは前に信頼していいって言われたサーバだ！」と判断してそのまま接続させてくれます。



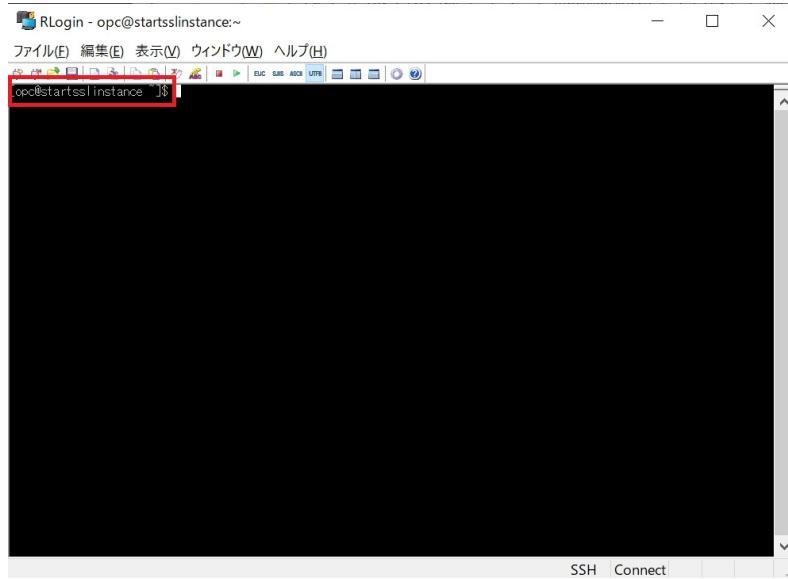
▲図 3.9 「公開鍵の確認」が表示されたら「接続する」をクリック

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 3.10）してください。



▲図 3.10 「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

「opc@startsslinstance」と表示（図 3.11）されたら無事サーバに入っています。SSH でのログイン成功、おめでとうございます！



▲図 3.11 「opc@startsslinstance」と表示されたら成功！

もし「opc@startsslinstance」と表示されず、代わりに「SSH2 User Auth Failure "publickey,ssh-rsa,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 3.12）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 3.12 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 3.13）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 3.13 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

3.1.2 Mac のターミナルを使ってサーバに入ってみよう

Mac を使っている方は、ターミナル（図 3.14）を起動してください。



▲図 3.14 最初からインストールされている「ターミナル」を使おう

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」という意味ですので、「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、これは「初めてに入るサーバだけど信頼していいですか？本当に接続しますか？」と聞かれていますので、「yes」と打ってReturnキーを押してください。するとMacはちゃんとこのサーバのことを覚えてくれて、次回以降は「これは前に信頼していいって言われたサーバだ！」と判断してそのまま接続させてくれます。

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

「opc@startsslinstance」と表示されたら無事サーバに入っています。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。

3.2 ターミナルでサーバを操作・設定してみよう

ようやくサーバに入れたので、ここからはターミナルの基本的な操作を試してみましょう。

3.2.1 プロンプトとは？

では黒い画面で何回かEnterキー（あるいはReturnキー）を押してみましょう。（図3.15）普通に改行されますよね。



▲図 3.15 Enterキーを押すと改行されて、プロンプトが常に表示されている

このとき左側にずっと出ている次のような表示は「プロンプト」といって、ログインしているユーザ名やサーバの名前などが表示されています。

```
[opc@startsslinstance ~]$
```

プロンプトを見るといまは「opc」という一般ユーザであることが分かります。これからサーバに色々な設定をしたいのですが、一般ユーザだと権限がないので「root」という全権限をもったユーザになりましょう。

```
$ sudo su -
```

と書いて Enter キーを押すと root になります。(図 3.16) 「\$」はプロンプトを表していますので入力しないでください。root になれたままた何回か Enter キーを押して改行してみましょう。



▲図 3.16 sudo su -を書いて Enter キーを押すと root になる

いちばん左側に出ているプロンプトが次のように変化しましたか？

```
[root@startsslinstance ~]#
```

ユーザ名が「opc」から「root」に変わりました。それからいちばん右の部分も「\$」から「#」に変わっています。プロンプトは**一般ユーザだと「\$」で全権を持っているrootだと「#」**という表示になります。今後は「このコマンドを root で実行してください」のように実行ユーザを詳しく書くことはしませんので、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実行、「#」だったら root で実行するんだ、と思ってください。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。

【コラム】ターミナルを閉じたいときは？

もう今日の勉強は終わり！ サーバとの接続を切ってターミナルを閉じたい、というときは exit (イグジット) というコマンドを叩きます。

```
# exit
```

root になっているときに exit を叩くと opc に戻れます。そして opc で再び exit を叩くと、サーバの接続を切ってターミナルを閉じることができます。

```
$ exit
```

exit をせずに右上の赤い×を押してウィンドウを閉じるのは、電話を切るときに通話オフのボタンを押さずに電話線を引っ張くような乱暴な切り方なのでお勧めしません。

3.3 NGINX をインストールしよう

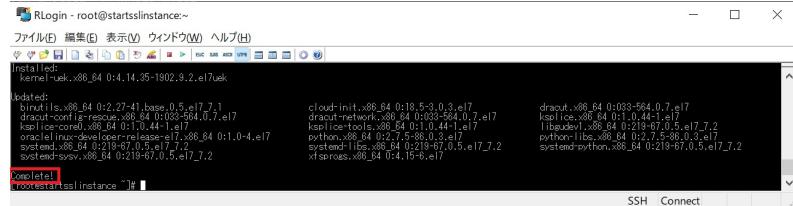
それでは必要なミドルウェアをインストールしていきましょう。最初に root になっておいてください。インストールするときは yum (ヤム) というコマンドを使います。

```
$ sudo su -
```

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。画面にたくさん文字が流れて、少し時間がかかりますが、最後に [Complete!] と表示されたら問題なく完了しています。(図 3.17) 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「これとこれを更新するけどいい？ 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て、y と入力して Enter キーを押さないと更新されません。

```
# yum update -y
```

3.3 NGINX をインストールしよう



▲図 3.17 最後に [Complete!] と表示されたらアップデートは完了

続いて NGINX^{*1}を入れます。2020 年 1 月現在の安定バージョン^{*2}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いくか）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.el7.2.noarch.rpm
# 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
```

*1 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/>

*2 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。

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他にも Apache をはじめとして色々な種類があります。

インストールが終ったので、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^{*3}で、NGINX の自動起動を有効 (enable) にします。

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

3.4 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^{*4}してみましょう。

```
# firewall-cmd --list-services
dhcpcv6-client ssh
```

dhcpcv6-client と ssh は通つていいけれど、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 を追加して、変更が反映されるよう再読み込みします。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
```

^{*3} 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

^{*4} 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

```
# 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、誰がファイアウォールを通してもらえるのか確認してみましょう。http と https が追加されていれば問題ありません。

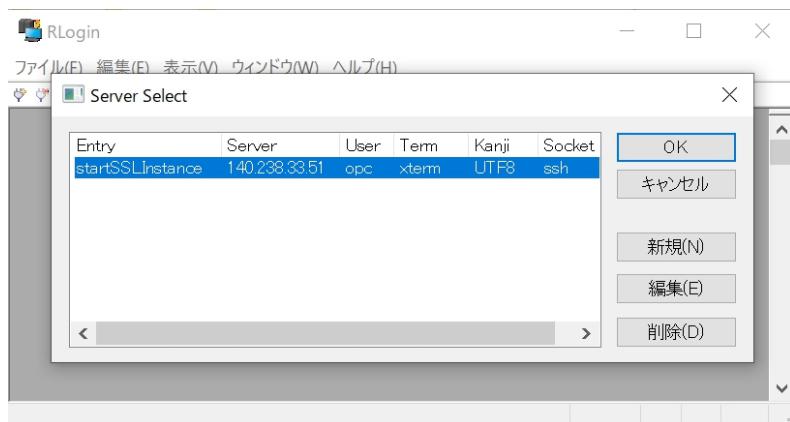
```
# firewall-cmd --list-services  
dhcpcv6-client http https ssh
```

3.5 OS を再起動してみよう

再起動時に NGINX がちゃんと自動起動してくるか確認するため、reboot（リブート）コマンドでサーバを再起動してみましょう。

```
# reboot
```

SSH の接続も切れてしまいますが、割とすぐに再起動しますので再度 RLogin やターミナルで接続（図 3.18）してみてください。今度はさっきと同じ設定でそのまま接続できるはずです。



▲図 3.18 さっきと同じ設定で接続してみよう

以上で「サーバを立てる」という作業はおしまいです。

3.5.1 ターミナルはなんのためにある？

ターミナルで yum や vi を叩いてサーバの設定を色々してきましたが、ここで「結局、ターミナルって何なの？」という振り返りをしておきましょう。

ターミナルはサーバを操作するための画面です。

皆さんが出でるときはモニタに表示された画面を見ながらキーボードとマウスを使って「フォルダを開いて先週作った Word ファイルを探す」とか「Word ファイルを開いて今週の報告書を書く」というような操作をするとと思います。フォルダを開くときは「ダブルクリック」をして、書いた内容を保存するときは「上書き保存する」ボタンを押しますよね。

サーバも同じです。サーバを使うときは「ターミナル」という画面を開いて操作します。ディレクトリ^{*5}を開いて移動するときはダブルクリックの代わりに cd^{*6}というコマンドを叩いて移動しますし、ディレクトリの中を見るときもダブルクリックでフォルダを開く代わりに ls コマンドを叩いて見ます。

皆さんが出でている Windows や Mac といった「パソコン」だったらマウスやキーボードを使ってアイコンやボタンを見ながら操作できますが、サーバは基本的にこの真っ黒な「ターミナル」で文字を打って操作します。パソコンのときはダブルクリックやボタンを押す、という形で伝えていた命令がすべてコマンドに置き換わっていると思ってください。

パソコンもないのにマウスやキーボードだけあっても意味が無いように、ターミナルもそれ単体では何もできません。操作対象であるサーバがあつて初めて役に立つ道具なのです。

ちなみにターミナルは背景の色も文字の色も好きに変えられます。どうしても「黒い画面怖い！」という感覚が抜けない人は、ピンクとかオレンジとか好きな色にしてみましょう。^{*7}

まとめるとターミナルとはサーバを操作するための画面で、操作するときにはコマンドという命令を使います。

^{*5} Linux ではフォルダのことをディレクトリと呼びます。

^{*6} change directory の略。

^{*7} Mac のターミナルはそもそも黒じゃなくて白ですね。

3.6 なぜか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、ブラウザで開いてみました。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 3.19）



▲図 3.19 なぜか HTTP でサイトが表示されない…

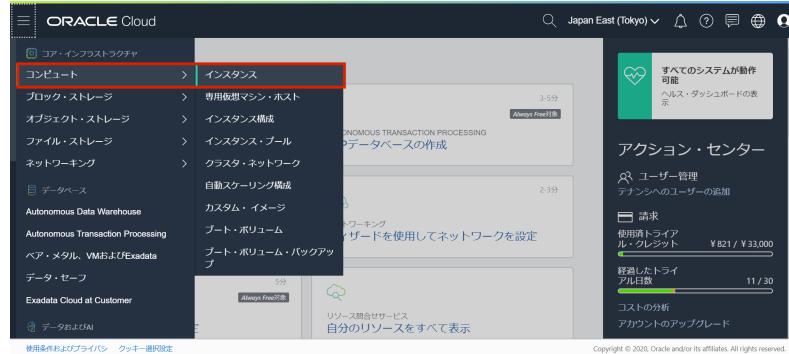
403 や 404 や、あるいは 502 などのステータスコードも返ってきていないので、そもそも NGINX までたどり着けていないようです。

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、サーバの外にももう 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「コンピュート」の「インスタンス」を開きます。（図 3.20）

第3章 ウェブサーバの設定をしよう



▲図 3.20 [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 3.21)



▲図 3.21 [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 3.22)

3.6 なぜかサイトが見られない

The screenshot shows the Oracle Cloud Infrastructure dashboard. On the left, there's a green bar indicating the instance is running. The main panel displays 'Instance Information' with various details like region, type, and network. Below it is the 'Virtual NIC Information' section. At the bottom right of this section, there's a link labeled 'Subnet' which is highlighted with a red box.

▲図 3.22 [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 3.23)

This screenshot shows the Oracle Cloud Infrastructure interface focusing on a specific route table. It displays details such as CIDR block, converter ID, and creation date. Below this, the 'startdns01 (Route) Component Subnets' section is shown, containing a table with one item. The 'Route Table' link at the top of this section is highlighted with a red box.

▲図 3.23 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 ~] をクリックします。(図 3.24) 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 ウェブサーバの設定をしよう

The screenshot shows the Oracle Cloud Infrastructure dashboard. In the top right, it says "Japan East (Tokyo)". Below that, there's a summary box with network details: CIDR ブロック: 10.0.0.0/24, 仮想ルーター MAC アドレス: 00:00:17:42:14:B4, サブネット・タイプ: リージョナル. To 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..., and ルート表: Default Route Table for VirtualCloudNetwork-20200120-2319. Below this is 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" (Enabled). The table has columns: 名前 (Name), 状態 (Status), コンバーティメント (Conversion), and 作成日 (Created Date). The item is listed as "Default Security List for VirtualCloudNetwork-20200120-2319" with status "使用可能" (Enabled), created by "starfids01 (ルート)" on 2020年1月20日(月) 14:55:54 UTC. A note below says "1アイテムを表示中 < ページ1 >". At the bottom, there are links for "使用条件およびプライバシー" and "Cookie 選択設定".

▲図 3.24 [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] をクリックします。(図 3.25)

The screenshot shows the Oracle Cloud Infrastructure dashboard. In the top right, it says "Japan East (Tokyo)". Below that, there's a table titled "イングレス・ルール" (Ingress Rules) with three items. The first item is highlighted with a red box around the "イングレス・ルールの追加" (Add Ingress Rule) button. The table has columns: ステータス (Status), ソース (Source), IPプロトコル (IP Protocol), ソース・ポート範囲 (Source Port Range),宛先ポート範囲 (Destination Port Range), タイプとコード (Type and Code), 許可 (Allow), and 説明 (Description). The first rule is for TCP port 22 from 0.0.0.0/0. The second rule is for ICMP port 3-4 from 0.0.0.0/0. A note on the right says "次に対するICMPトライアッカ: 3, 4宛先に到達できません: フラグメントデリケーションが必要ですが、フラグメントルミングがオフです。" At the bottom, there are links for "使用条件およびプライバシー" and "Cookie 選択設定".

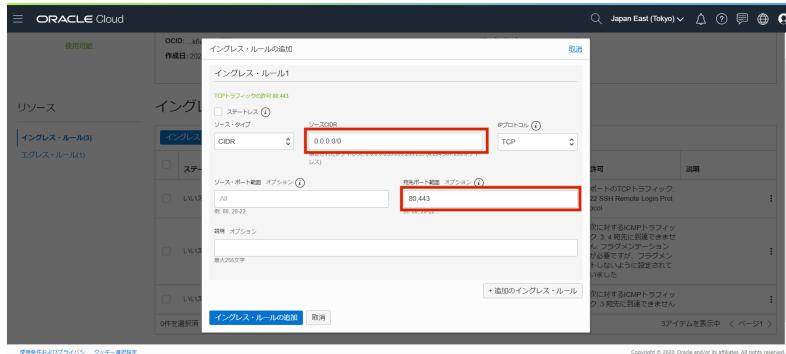
▲図 3.25 [イングレス・ルールの追加] をクリック

[ソース CIDR] に [0.0.0.0/0]^{*8}を入力します。[宛先ポート範囲] には [80,443]^{*9}を入力します。(図 3.26) 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。

^{*8} 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れます、ということですね

^{*9} ポート番号とは、サーバという家や、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 なぜかサイトが見られない



▲図 3.26 [ソース CIDR] に [0.0.0.0/0]、[宛先ポート範囲] には [80,443] を入力

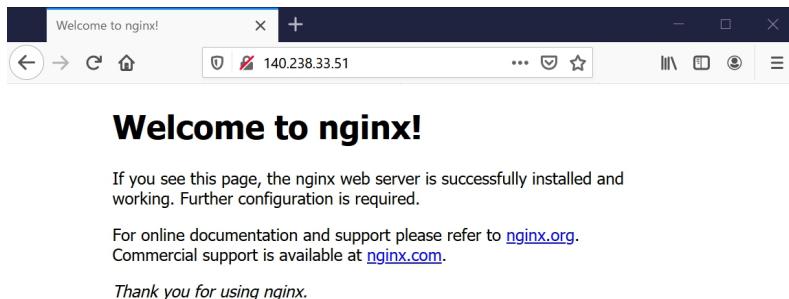
[イングレス・ルール] に、HTTP（80 番ポート）と HTTPS（443 番ポート）へのリクエストを通す設定が追加されました。（図 3.27）

ステータス	ソース	IPプロトコル	ソース・ポート範囲	宛先ポート範囲	タイプとコード	許可	説明
いいえ	0.0.0.0/0	TCP	All	22	ポートのTCPトラフィック SSH Remote Login Protocol	○	次に対するTCPトラフィック 3, 4 種元に到達できませ ん。 フラグメントーション コラージュ、アグリゲ ートドしないように設定さ れていました。
いいえ	0.0.0.0/0	ICMP			3, 4	○	次に対するICMPトラフィック 3, 4 種元に到達できませ ん。
いいえ	10.0.0.0/16	ICMP			3	○	次に対するICMPトラフィック 3 種元に到達できませ ん。
いいえ	0.0.0.0/0	TCP	All	80	ポートのTCPトラフィック 80	○	
いいえ	0.0.0.0/0	TCP	All	443	ポートのTCPトラフィック 443 HTTPS	○	

▲図 3.27 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 3.28）



▲図 3.28 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」はクリアです。

3.7 ドメイン名の設定をしよう

ウェブサーバの準備ができたので、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！」という人は、先に「DNS をはじめよう」*10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。

あなたが買ったドメイン名を使って「ssl. 自分のドメイン名」の A レコードを作成して、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 3.29）

ホスト名	TYPE	TTL	VALUE	優先	状態	追加
ssl .startdns.fun	A ▾	3600	140.238.33.51	51	有効 ▾	追加

▲図 3.29 「ssl. 自分のドメイン名」の A レコードを作成する

*10 <https://mochikoastech.booth.pm/>

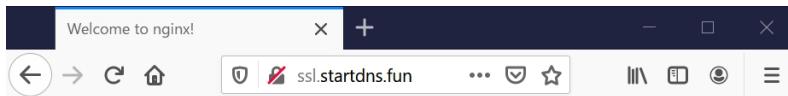
A レコードを追加できたかどうかは、次の dig コマンドで確認できます。dig コマンドをたたいた結果、サーバの IP アドレスが返ってくれば A レコードは設定できています。

```
$ 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」を叩いてみましょう。先ほどと同じ NGINX のページが表示されるはずです。（図 3.30）



Welcome to nginx!

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

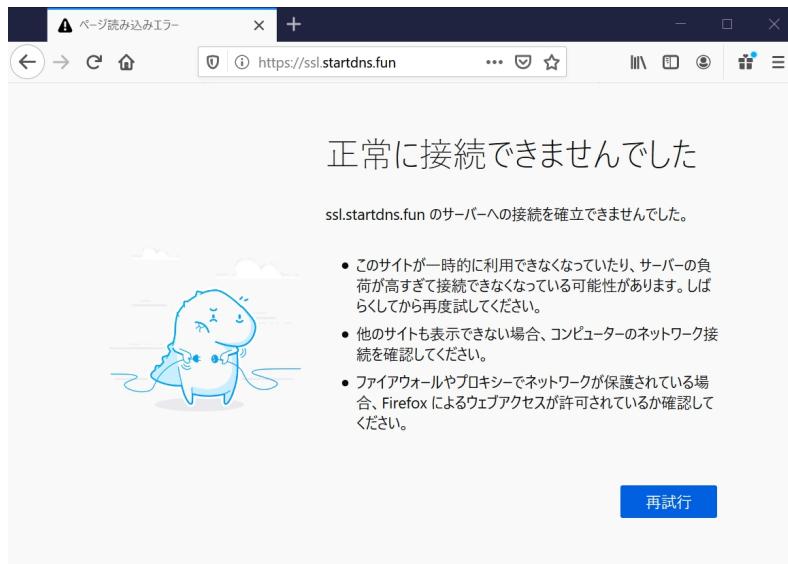
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

▲図 3.30 「ssl. 自分のドメイン名」で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が、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 3.31）されました。

第3章 ウェブサーバの設定をしよう



▲図 3.31 HTTPS で開くと [正常に接続できませんでした] と表示された

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 証明書を取得しよう

ウェブサーバとドメイン名を準備したら HTTP でサイトが見られました。
今度は HTTPS で見るために、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、関わってくる登場人物が多いので、最初に登場人物一覧をご紹介します。

- 秘密鍵 (startssl.key)
- CSR (startssl.csr)
- SSL 証明書 (server.crt)
- 中間 CA 証明書 (ca-bundle.ca)
- SSL 証明書+中間 CA 証明書 (startssl.crt)

これらの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っておきます。^{*1}

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

4.2 秘密鍵 (startssl.key) を作ろう

先ず最初に作成するのが秘密鍵です。秘密鍵は「.key」や「.pem」^{*2}という拡張子で作成されることが多いです。秘密鍵は名前のとおり「秘密」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。決してメールに添付して送ったり、サーバ内で誰でも見られる /tmp/ 以下に置いたりしてはいけません。

次の openssl コマンドを叩いて、秘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 の秘密鍵を作る。作った秘密鍵は /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 ビットで。」という意味です。できあがった秘密鍵は、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
.....
```

^{*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で `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 になれます

^{*2} 秘密鍵が 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

```
.....+++  
e is 65537 (0x10001)  
  
# chmod 600 /etc/nginx/ssl/startssl.key
```

できあがった秘密鍵を、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/hq0jqLFJUTbfg9EP8w+JVW4+8hp40Sklymc  
NRcvPYUBQy3wK+w527rksodkGZ77c6Q+XxRtH/wpo3H+xwhmJvi+T2o=  
-----END RSA PRIVATE KEY-----
```

【コラム】SSL 証明書の秘密鍵にパスフレーズは設定すべき？

openssl コマンドで秘密鍵を作るとき、-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、指定の暗号で暗号化された秘密鍵が出力されます。^{*3} [Enter pass phrase:] や [Verifying - Enter pass phrase:] と表示された際に、いくらキーボードを叩いて入力しても、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」を入力  
Verifying - Enter pass phrase: ←もう一度「startssl」を入力
```

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ていると、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。

これを回避するには、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」とリクエストする、申請書のようなものです。

左から順に、「opensslコマンドで、CSRを、新しく、秘密鍵はstartssl.keyで、startssl.csrというファイル名で作って」という意味です。

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

*3 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

4.3 CSR (`startssl.csr`) を作ろう

CSR を作成するため、いくつか質問をされます。(表 4.1)*4

▼表 4.1 CSR 作成時に聞かれる質問

必須/任意	質問	説明	入力する内容
必須	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
任意	Organizational Unit Name	組織単位名 (OU)	入力なし
必須	Common Name	コモンネーム (CN)	ssl.startdns.fun

```
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。「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しょう。

```
# 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
```

CSR を cat コマンドで表示して、「-----BEGIN CERTIFICATE REQUEST-----」から「-----END CERTIFICATE REQUEST-----」までをメモしておきましょう。メモした CSR はこの後使用します。

*4 本著では 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

```
# cat /etc/nginx/ssl/startssl.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICqzCCAZMCAQAwZjELMAkGA1UEBhMCS1AxDjAMBgNVBAgMBVRva3lvMRQwEgYD
VQQHDAtTaGluanVrdS1rdTEWMBQGA1UECgwNbW9jaG1rb0FzVGVjaDEZMBcGA1UE
(中略)
jP1RMQS3PuYDE6QIVJ5zbMC+RIydSQ/0Dr9VUHWiYqDPjx+BpphYT5AxMwbw9/m5
noKGvJl1Mt7G03Awa/TX
-----END CERTIFICATE REQUEST-----
```

【コラム】openssl コマンドのユーティリティ（道具）たち

第5章「SSL/TLSについて学ぼう」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。秘密鍵を作るときには genrsa、CSR を作るときには req というユーティリティを、openssl コマンドの引数につけて実行しましたよね。

```
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

パスフレーズありの秘密鍵から、パスフレーズなしの秘密鍵を作るとときは 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^{*5}で確認できます。

4.3 CSR (`startssl.csr`) を作ろう

```
$ 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
```

答え _____

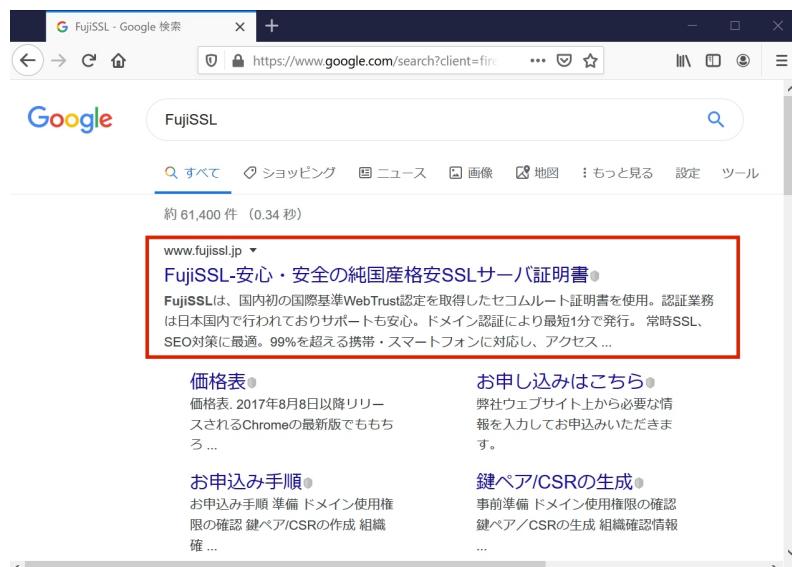
解答

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（クライアント）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、エンドユーザが「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。

4.4 SSL 証明書の取得申請を出そう

[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。^{*7}



▲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

^{*7} 検索すると色々な種類の 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 について学ぼう」で後述します

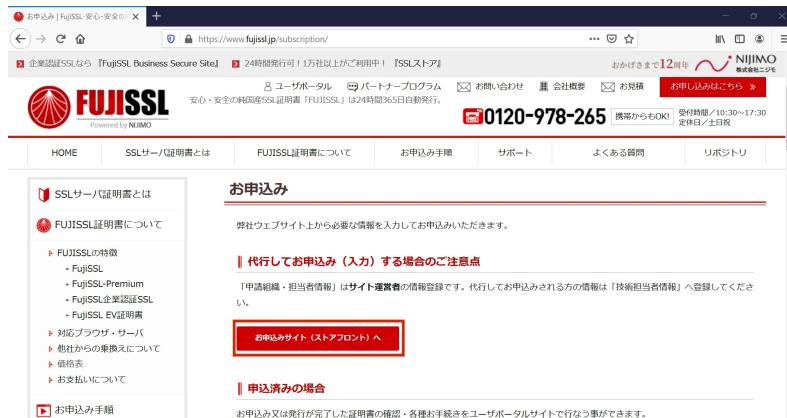
4.4 SSL 証明書の取得申請を出そう

右上の「お申し込みはこちら」をクリック（図 4.2）します。



▲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

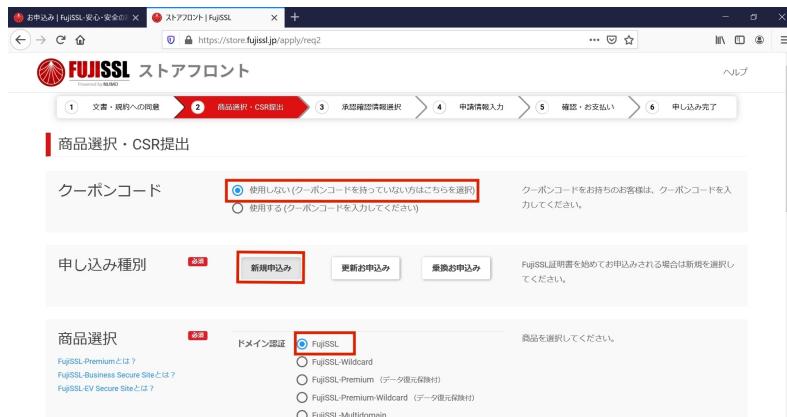
指定された文書と「収集した個人情報の利用目的」を確認した上で、チェックボックスにチェックを入れて「次へ」をクリック（図 4.4）します。

第4章 SSL証明書を取得しよう



▲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「新規申し込み」、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。



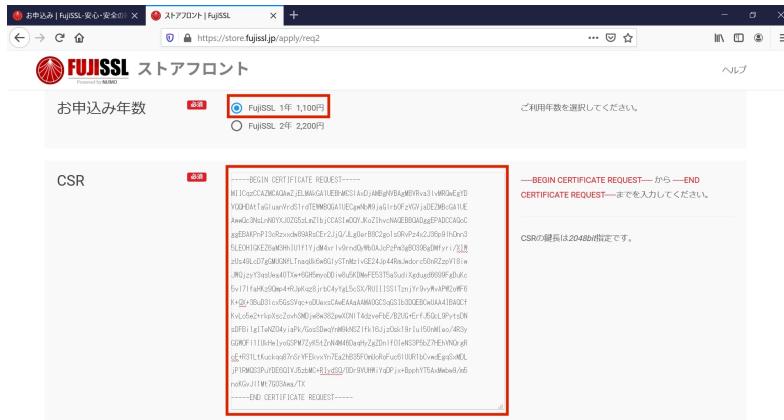
▲図4.5 クーポンコード、申し込み種別、商品選択を確認

下へスクロールして、お申込み年数が「FujisSL 1年 1,100円」*8になっていることを確認（図4.6）したら、CSRに、先ほど「-----BEGIN CERTIFICATE REQUEST-----」から「-----END CERTIFICATE REQUEST-----」までをメモしておいたCSRをペース

*8 2019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

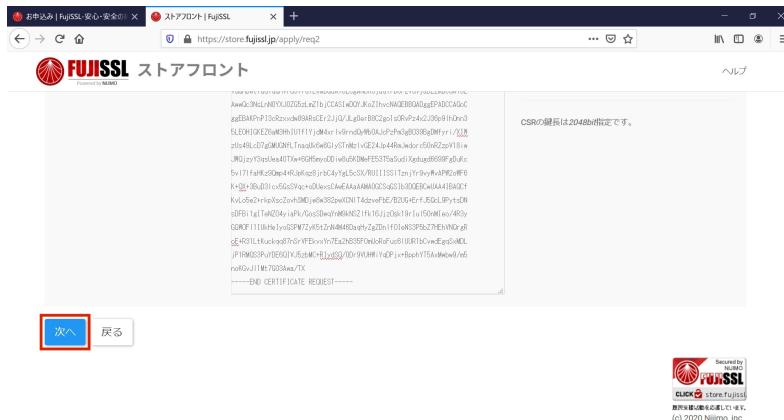
4.4 SSL 証明書の取得申請を出そう

トします。



▲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。



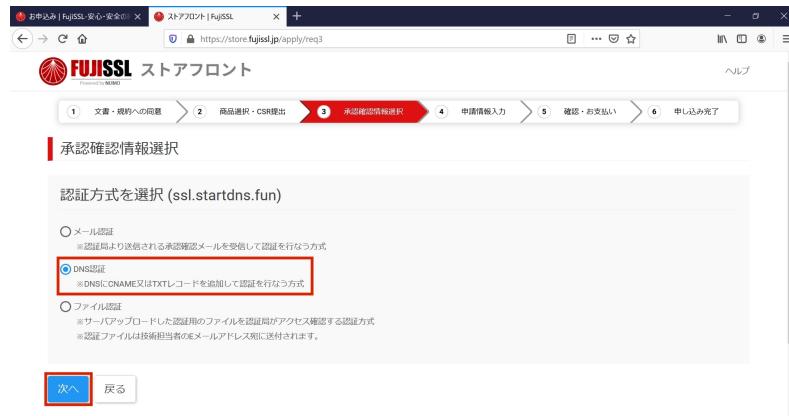
▲図 4.7 [次へ] をクリック

認証方式は「DNS 認証」を選択（図 4.8）してください。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 を踏むか、指定された内容のリソースレコードを DNS で追加するか、サイトに指定された内容のファイルをアップするか、いずれかの方法で「このドメイン名（筆者なら ssl.startdns.fun）は

第4章 SSL証明書を取得しよう

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）

A screenshot of the Fujissl application interface showing the '申請情報入力' (Application Information Input) screen. The URL is https://store.fujissl.jp/apply/req4. The page shows a six-step process: 1. 文書・契約への依頼, 2. 商品選択・CSR提出, 3. 承認確認情報選択, 4. 申請情報入力 (highlighted in red), 5. 確認・お支払い, 6. 申し込み完了. A table titled 'お申し込み情報' (Application Information) lists the following details: 商品名 (Product Name): Fujissl; 年数 (Term): 1年; 種別 (Type): 新規; 金額 (Amount): 1,100円. A table titled '証明書情報' (Certificate Information) lists the following details: コモンネーム (Common Name): ssl.startdns.fun (status: 反映されます.); 相続名 (Inheritor Name): mochikooAsTech (status: 反映されません.); 部署・部局 (Department): 新宿 (status: 反映されません.); 郵便番号 (Postal Code): Tokyo (status: 反映されません.); 市区町村 (City/Town/Village): Shinjuku ku (status: 反映されません.); 國 (Country): JP (status: 反映されません.). Below these tables is a section titled '申請組織担当者情報' (Information of the organization responsible for the application). The '申請企業・団体名' (Name of the application company/group) field contains '例) NIJIMO, INC.' and has a '必須' (Required) status indicator. There is also a 'ヘルプ' (Help) button.

▲図4.9 [申請情報入力]画面

4.4 SSL 証明書の取得申請を出そう

【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、個人名などは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。^{*9}この後、ここで入力した【E メールアドレス】宛てに、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。

The screenshot shows the Fujissl application interface. The form is titled '申請組織担当者情報' (Organization Contact Information). It contains several input fields with placeholder text and examples:

- 申請企業・団体名: 'mochikoAsTech' (Placeholder: 例) NUJIMO, INC. (Note: If individual, enter name)
- お名前: '姓: Nekomura, 名: Mochiko' (Placeholder: 例) 姓: Fuji 名: Taro
- 役職: 'Technical Writer' (Placeholder: 例) CEO
- 国名: '日本 (Japan)' (Placeholder: 例) Tokyo
- 郵便番号: '157-0071' (Placeholder: 例) 000-0000
- 所在地: 'Tokyo, Shinjuku-ku, 4-1-6 Shinjuku' (Placeholder: 例) Shibuya-ku, 1-12-2, Shibuya, CROSS OFFICE SHIBUYA 5F

▲図 4.10 【申請組織担当者情報】と【技術担当者情報】を入力

The screenshot shows the Fujissl application interface with the '次へ' (Next) button highlighted in red. The form contains the same fields as in Figure 4.10, with the following additional information:

- 所在地: 'Tokyo, Shinjuku-ku, 4-1-6 Shinjuku, SHINJUKU MIRAINA TOWER' (Placeholder: 例) Tokyo, Shibuya-ku, 1-2-2, Shibuya, CROSS OFFICE SHIBUYA 5F
- 電話番号: '03-xxxx-xxxx' (Placeholder: 例) 03-0000-0000
- Eメールアドレス: 'startdns.01@gmail.com' (Placeholder: 例) taro@fujissl.jp (Note: Certificate delivery address)

▲図 4.11 【次へ】をクリック

*9 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。詳細については後述します

第4章 SSL証明書を取得しよう

[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 'Payment Information Input' section of the Fujissl storefront. It includes fields for 'Payment Amount' (1,100 yen), 'Card Type' (JCB, MasterCard, VISA, American Express, Diners Club, etc.), and credit card details (number, expiration date, security code, name). The card number field is highlighted with a red box.

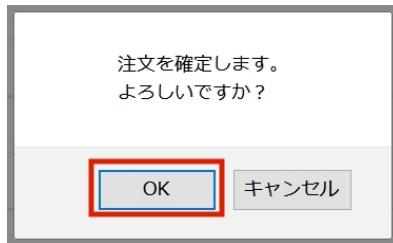
▲図4.12 クレジットカード情報を入力

[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。

The screenshot shows the 'Document Delivery Address' section of the Fujissl storefront. It lists three address fields: 'Invoice Recipient Name' (mochikoAsTech), 'Delivery Recipient Name' (mochikoAsTech), and 'Receipt Recipient Name' (mochikoAsTech). The 'Invoice Recipient Name' field is highlighted with a red box. At the bottom, there is a button labeled 'Confirm and Proceed' (上記の内容で注文を確定する) and a 'Back' button.

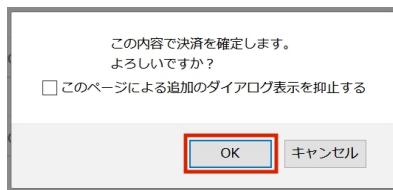
▲図4.13 [上記の内容で注文を確定する] をクリック

「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^{*10}をしたら、[OK] をクリックします。



▲図 4.15 1,100 円払うケツイをして [OK] をクリック

お申し込み完了のページが表示（図 4.16）されました。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。

^{*10} 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

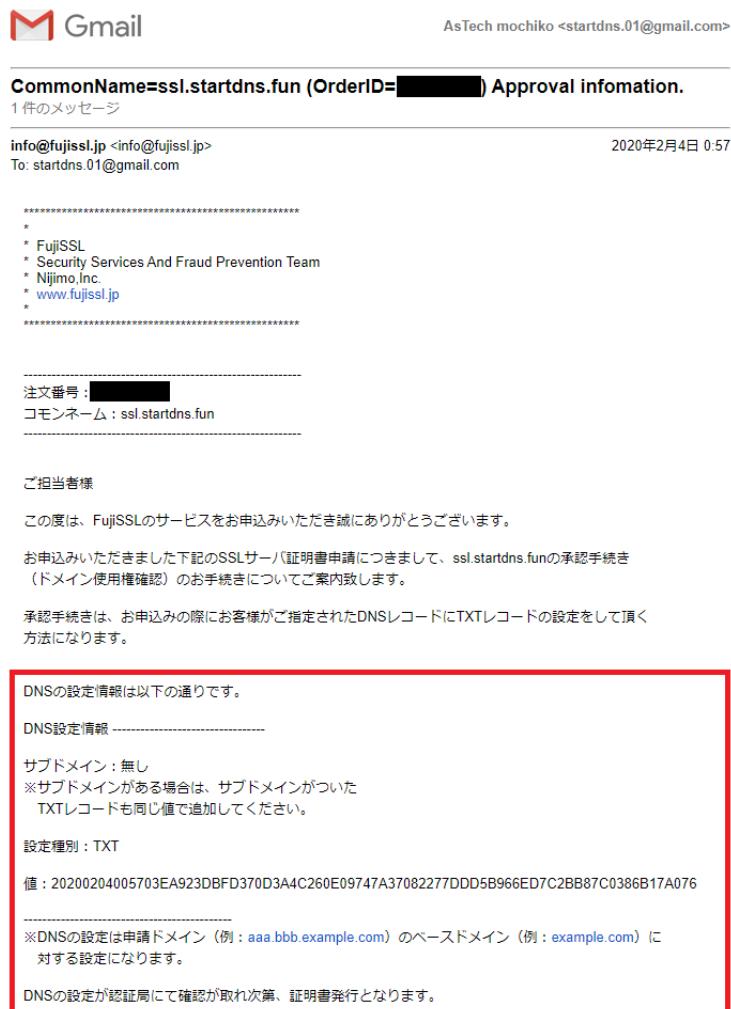
第4章 SSL証明書を取得しよう



▲図4.16 [お申込み完了]画面

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）がすぐに届きました。「CSRのコモンネームで指定したドメイン名がssl.startdns.funの場合、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 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ペーストします。

第4章 SSL 証明書を取得しよう

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	追加

▲図 4.18 「自分のドメイン名」の TXT レコードを追加する

TXT レコードができたかどうかは、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 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  
"20200204005703EA923DBFD370D3A4C260E09747A37082277DDD5B966ED7C2BB87C0386B17A076"
```

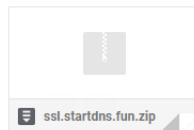
TXT レコードを追加してから、おおよそ 30 分後に SSL 証明書がメール（図 4.19）で届きました。



▲図 4.19 SSL 証明書がメールで届いた

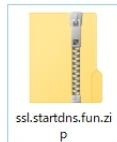
メールに添付されている ZIP ファイル (ssl_自分のドメイン名.zip) に SSL 証明書が

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

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)



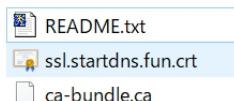
▲図 4.22 右クリックして [すべて展開] をクリック

[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 [server.crt] が SSL 証明書で、[ca-bundle.ca] が中間 CA 証明書です。README はファイルの説明書です。



▲図 4.24 server.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、この 2 つのファイルをサーバにアップロードしましょう。

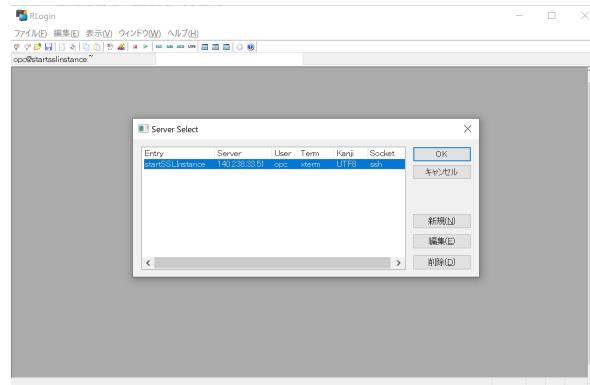
4.6 SSL 証明書をサーバに設置しよう

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。

4.6.1 Windows で証明書と中間 CA 証明書をアップしよう

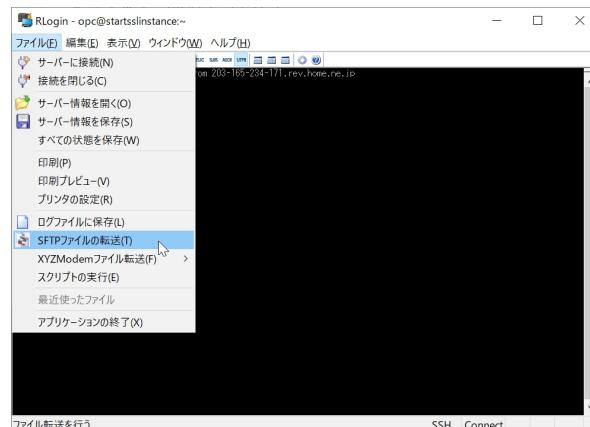
Windows のパソコンを使っている方は、RLogin を起動します。[startSSLInstance] を選択して、[OK] をクリック（図 4.25）します。

4.6 SSL 証明書をサーバに設置しよう



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

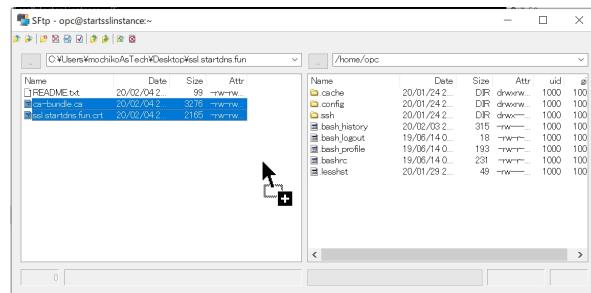
黒い画面が開いたら、[ファイル] から [SFTP ファイルの転送] をクリック（図 4.26）します。



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

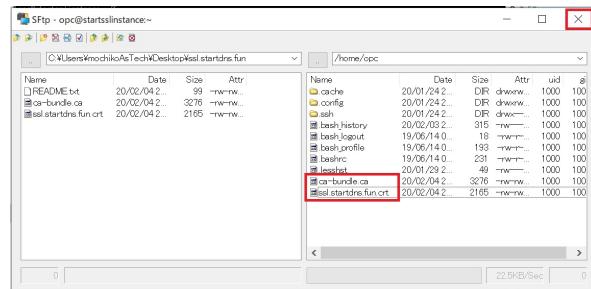
左側があなたのパソコンで、右側がサーバです。左側をデスクトップに展開したフォルダ^{*11}にしたら、[server.crt] と [ca-bundle.ca] を右側にドラッグ&ドロップ（図 4.27）してください。これでサーバの「/home/opc」にファイルがアップロードされます。

^{*11} 筆者の場合は「C:\Users\mochikoAsTech\Desktop\ssl.startdns.fun」でした



▲図 4.27 [server.crt] と [ca-bundle.ca] を右側にドラッグ&ドロップ

右側に [server.crt] と [ca-bundle.ca] がアップされたら、×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)



▲図 4.28 [server.crt] と [ca-bundle.ca] がサーバにアップされた

4.6.2 Macで証明書と中間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。scp コマンドを使って、Macの中にある [server.crt] と [ca-bundle.ca] を、サーバにアップロードします。[ZIPを展開したフォルダ] と [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 server.crt ca-bundle.ca opc@パブリック IP アドレス:/home/opc/
server.crt    100%   0    0.0KB/s   00:00
ca-bundle.ca  100%   0    0.0KB/s   00:00
```

4.6.3 証明書を 1 ファイル（startssl.crt）にまとめよう

サーバにログインしている状態で、次のコマンドを叩いて、先ほどアップロードした [server.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す。

```
$ sudo su -  
# ls /home/opc/  
ca-bundle.ca  ssl.startdns.fun.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。^{*12}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.startdns.fun.crt /home/opc/ca-bundle.ca > startssl.crt
```

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^{*13}です。

```
# cat /etc/nginx/ssl/startssl.crt  
-----BEGIN CERTIFICATE-----  
MIIF/DCCB0SgAwIBAgIQaoS/P1b4mCR8mn5/WUrI6zANBgkqhkiG9wOBAQsFADbN  
MQswCQYDVQQGEwJKUDE1MCMGA1UEChMcU0VDT0OgVHJ1c3QgU3lzdGVtcyBDTy4s  
(中略)  
31pTIUPabkFxDPjs1Vw9c7z3Vgk2fnpuwE21rE+46zrJ3oTRqsABDbYreK1a5vsG  
tcnpU1L1hrk/rC3JuI2ttHVaHU+1JCgTSRpv03a44azDy15T5C97dGxuowPgcaMQ  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
MIIEpzCCA4+gAwIBAgIJIRmxVPM8X14AMA0GCSqGSIb3DQEBCwUAMF0xCzAJBgNV  
BAYTAkpQMSUwIwYDVQQKExxTRUNPTSBUcnVzdCBTeXNOZW1zIENPLixMVEQuMSCw  
(中略)  
g3tiJA1FIpfXXD4cArZo6Z1XJ26B4H7vk5GmyR6poDy/CRvC7VIz3xp6o2348W1j
```

^{*12} 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 コマンドを使っています

^{*13} 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

```
32S9pEuZhtxtMvPjnsHIWPNdz8pHv21x7bYwDnocwN2uk3Qrr1jxTQ9evg==  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
MIIEcjCCA1qgAwIBAgIJERmw+nLg2EjGMAOGCSqGSIB3DQEBCwUAMFAxCzAJBgNV  
BAYTAkpQMrgwFgYDVQQKEw9TRUNPTSBucnVzdC5uZXQxJzA1BgNVBAstH1N1Y3Vy  
(中略)  
u5ZuCjxerxj3qS1rM46bcEfjopnaD7hnJXSYiL1d0yw5zSW2PEe+LHdoIAb2I6D8  
8UFJHOCl6sY518jhjk0Os1yeu1C/RcYO+NHKZkFEeEb6ezOsg=  
-----END CERTIFICATE-----
```

これで証明書ファイルの準備は完了です。

4.7 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で、もともとあった設定ファイルを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。

```
# vi startssl.conf  
  
server {  
    listen 80 default_server;  
    return 301 https://$host$request_uri;  
}  
  
server {  
    listen      443 ssl http2;  
    server_name  ssl.startdns.fun;  
  
    # 秘密鍵  
    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  
    ssl_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AE  
  
    # 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;
    index index.html index.htm;
}

```

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^{*14}で確認できます。

クライアントが希望する暗号スイート群をサーバに送ったら、サーバ側がリストの上位から順に探していく、共通のものが見つかればその暗号スイートに決定します。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 [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。] と表示されてしまいます。(図 4.29)



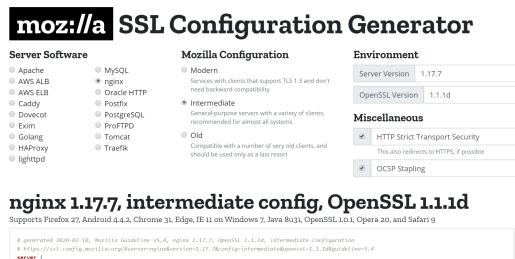
▲図 4.29 共通の暗号スイートが見つからず、接続できなかった様子

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、安全性と可用性（対応端末の多さ）のバランスを考えて決めていく

^{*14} Transport Layer Security (TLS) Parameters <https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml>

ことになります。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、IPA^{*15}が公開している「SSL/TLS暗号設定ガイドライン^{*16}」で、「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つが提示されているので、まずはこれを参考にしながら、るべき設定を考えていく、という方法もあります。

他にも、Mozilla^{*17}が提供する「Mozilla SSL Configuration Generator^{*18}」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。こちらを参考にするのもよいでしょう。



▲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。

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

それからアクセスしたときに表示される、index.htmlの文字も「Let's start SSL/TLS」に変えておきましょう。

^{*15} 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

^{*16} 本著では「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

^{*17} ブラウザのFirefoxを作っているあのMozilla

^{*18} <https://ssl-config.mozilla.org/>

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

4.8 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 の再起動もしたのでブラウザで「http://ssl.自分のドメイン名」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！

【コラム】ロードバランサでも SSL ターミネーションできる

なお本著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 証明書を設置して、SSL ターミネーションを行う方法もあります。その場合、HTTPS で通信するのはエンド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^{*19}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ね。

さらに AWS の ELB^{*20}と ACM^{*21}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、本著では「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。

^s ちなみに本著で扱った FujiSSL 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

^{*20} Elastic Load Balancing の略。AWS のロードバランサ

^{*21} 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について学ぼう

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。この章では実践と照らし合わせながら SSL/TLS について学んでいきます。

5.1 SSL/TLSとは？

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殆ど²です。

5.1.1 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RFC7568⁴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。

本著ではSSL/TLSのことを指して、SSLという言葉を使用しています。

5.1.2 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、通信を保護するプロトコルがHTTPS(HTTP over SSL/TLS)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

*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。手紙の例と同じように、インターネットで通信を行う際、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 SSL証明書を取得するために、秘密鍵やCSRを作ったときのコマンドも、opensslコマンドでしたね

*³ 脆弱性（ぜいじゃくせい）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>

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データを安全に送受信できるようになります。

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。

5.2 「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？

本著では、「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^{*6}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いきました。

大手サイトが次々と HTTPS 化をはじめたのが、2016 年から 2017 年ごろだったと記憶しています。たとえばクックパッドは 2017 年 1 月^{*7}、日経新聞電子版は 2017 年 8 月^{*8}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。

5.3 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も、HTTPS 化し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「アセトネデブ」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://techlife.cookpad.com/entry/2017/04/19/190901>

^{*8} <https://www.nikkei.com/topic/20170808.html>

リットがあります。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。

5.4 HTTPのままだと起きるデメリット

ですが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。

5.4.1 サイトが「安全でない」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5.1）されます。

① 保護されていない通信 | ikinaristea.com/home/

▲図5.1 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5.2）を行っており、サイトをHTTPS化しないことで、エンドユーザにサイトが「安全でない」と思われるてしまう状況にあります。



▲図5.2 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

5.4.2 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

Amazonなどのサイトを開くと、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ー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、誰でも使える Wi-Fi スポットに、スマート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。

5.4.3 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 のままでいる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^{*9}しています。

^{*9} 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

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。

5.4.4 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。

5.5 HTTPS化すると得られるメリット

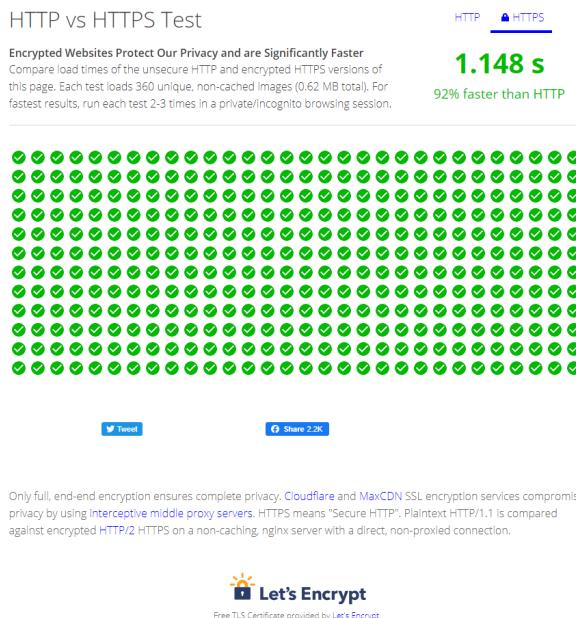
HTTPのままでいると起きるデメリットだけでなく、HTTPS化すると得られるメリットももちろんあります。

5.5.1 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、もはや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。^{*10}

^{*10} 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」を使う話も、最近はほぼ聞かなくなりました

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^{*11}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ま、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。たと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 5.3)^{*12}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 5.3 HTTP vs HTTPS Test

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、さらなる速度改善が見込まれます。

^{*11} 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

^{*12} HTTP vs HTTPS Test <https://www.httpvshttps.com/>

5.5.2 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^{*13}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^{*14}。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。

5.5.3 重要情報のアタリが付けにくくなる

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殆どが平文な中で、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アタリが付けにくくなります。大事なものも大事でないものもすべて同じ頑丈なアタッシュケースにしまって運べば、泥棒がどのアタッシュケースを狙えばいいのか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、HTTPSにすると得られるメリット、そ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。

5.6 SSL証明書とは

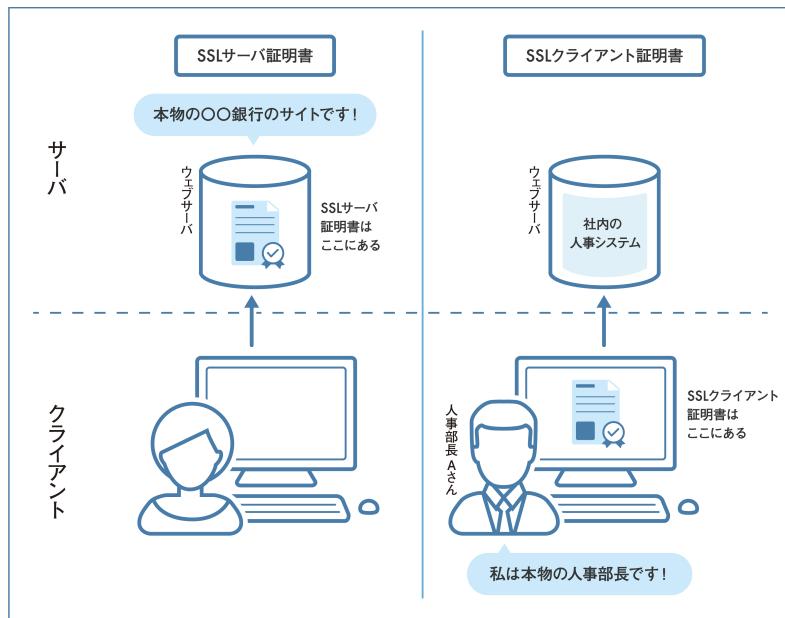
続いて、HTTPS化するため必要なSSL証明書について学んでいきましょう。

^{*13} <https://developers-jp.blogspot.com/2019/11/cookie-samesitenone-secure.html>,
<https://developers-jp.blogspot.com/2020/02/2020-2-samesite-cookie.html>

^{*14} 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

5.6.1 SSL サーバ証明書と SSL クライアント証明書

皆さんのがさっく取得、設置した「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 5.4）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。

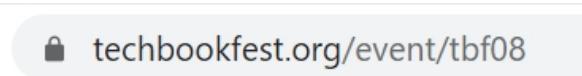


▲図 5.4 SSL サーバ証明書と SSL クライアント証明書

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」のことを指すことが殆どです。本著でも 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。

5.6.2 SSL 証明書はどんな場面で使われている？

SSL 証明書はどんな場面で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 5.5、図 5.6）が表示されているときに使われています。



▲図 5.5 Chrome で HTTPS のサイトを開いたとき



▲図 5.6 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 5.7、図 5.8）が表示されているときは使われていません。



▲図 5.7 Chrome で HTTP のサイトを開いたとき



▲図 5.8 Firefox で HTTP のサイトを開いたとき

5.7 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。

実は、SSL 証明書は異なる 3 つの仕事をしています。たとえば、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/>) に接続したとき、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、ページを見せて」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なく、暗号化に必要な鍵交換を行なう過程で SSL 証明書が使われます。

5.7.1 HTTPS の実際の流れ

では HTTPS の全体を流れを掴むため、あなたがさっそく作ったばかりの自分のサイト (<https://ssl.自分のドメイン名/>) を開いたときの、ざっくりとした手順を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^{*15}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
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）

^{*15} 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

- 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^{*16}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 3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 に記載されている FQDN^{*17}と、リクエスト先の FQDN (bank.example.com) 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^{*18}を行ないます。

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）とはまた別物
 - 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

^{*16} 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、解説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

^{*17} 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

^{*18} Diffie-Hellman 鍵交換。TLS1.3 からは

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。

5.7.2 ウェブページは 1 往復で表示されるわけじゃない

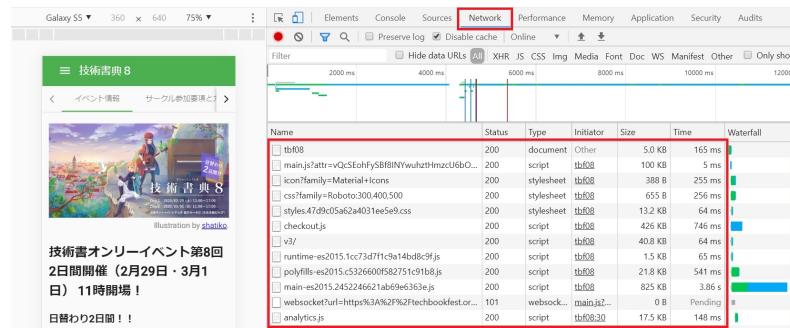
ところで 1 つのウェブページを表示するとき、クライアント（あなたのブラウザ）とサーバのやりとりは、「トップページをください」「はい完成品をどうぞ」の一往復だけではありません。次のように、何往復ものリクエストとレスポンスでパートを揃えていくってページが表示されます。

- 「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、たとえば技術書典 8 のサイト^{*19}を開くと、次のようにたくさんの行が表示（図 5.9）されます。2019 年 2 月時点、このページは 54 往復のリ

^{*19} <https://techbookfest.org/event/tbf08>

クエストとレスポンスで表示されており、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。

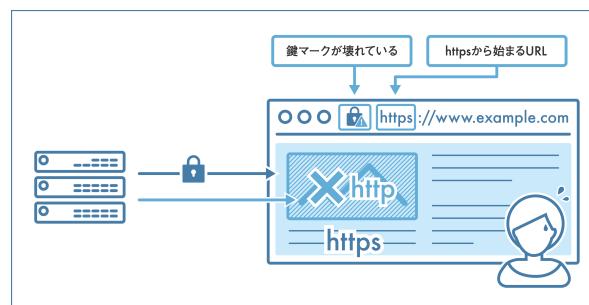


▲図 5.9 54 往復のリクエストとレスポンスで表示された技術書典 8 のページ

5.7.3 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、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 5.10）



▲図 5.10 混在コンテンツ（mixed content）はエラーが出る

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 5.11) を出すのです。



▲図 5.11 [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^{*20}からは、この混在コンテンツが存在した場合、対象のプロトコルが 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:// で始まる形式にしていた 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、たとえば「」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、

^{*20} <https://developers-jp.googleblog.com/2019/11/https.html>

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^{*21}」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、「」のように指定するのがいいでしょう。

5.7.4 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」^{*22}を使って、あなたが作ったサイトの SSL 証明書を確認（図 5.12）してみましょう。[FQDN1] に [ssl. 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 5.12 [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 5.13）されました。

^{*21} <https://google.github.io/styleguide/htmlcssguide.html#Protocol>

^{*22} https://sstool.cybertrust.ne.jp/support_tool/index01.php

5.7 SSL 証明書は異なる 3 つの仕事をしている

設定確認結果
証明書は正しく設定されています。

<証明書階層>

サーバ証明書

コモンネーム (CN)	ssl.startdns.fun
有効期間(JST)	2020/02/04 01:19:36 ~ 2021/02/05 23:59:59
▲ 詳細情報を非表示にする ▾ 証明書を文字列で表示する	
組織単位名 (OU)	
組織名 (O)	SECOM Trust Systems CO.,LTD.
市町村名 (L)	-
都道府県名 (S/ST)	-
国名 (C)	JP
公開鍵長	2048 bit
シリアル番号	6A84BF3F56F898247C9A7E7F594AC8EB
署名アルゴリズム	sha256WithRSAEncryption
Certificate Transparency	対応
コモンネーム (発行者) (CN)	FujiSSL Public Validation Authority - G3
Subject Alternative Name(SANs)	
DNS Name	ssl.startdns.fun

▼

中間CA証明書1

コモンネーム (CN)	FujiSSL Public Validation Authority - G3
▲ 詳細情報を非表示にする ▾ 証明書を文字列で表示する	
有効期間(JST)	2018/08/22 16:41:02 ~ 2028/08/22 16:41:02
公開鍵長	2048 bit
シリアル番号	22b9b154f33c5e5e00
署名アルゴリズム	sha256WithRSAEncryption
コモンネーム (発行者) (OU)	Security Communication RootCA2

▼

中間CA証明書2 (クロスルート)

コモンネーム (OU)	Security Communication RootCA2
▲ 詳細情報を非表示にする ▾ 証明書を文字列で表示する	
有効期間(JST)	2015/03/24 11:22:35 ~ 2023/09/29 11:22:35
公開鍵長	2048 bit
シリアル番号	12b9b0fa72e0d848c6
署名アルゴリズム	sha256WithRSAEncryption
コモンネーム (発行者) (OU)	Security Communication RootCA1

▲図 5.13 証明書の階層が表示された

[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。いろいろなことが記載されていますが、要約すれば次の表に挙げた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。ですがあなたは身も知らない D さんこと [Security Communication RootCA1] を信頼できますか？ D さんの正当性はいったい誰が証明するのでしょうか？

5.7.5 ルート証明書はトラストストアにある

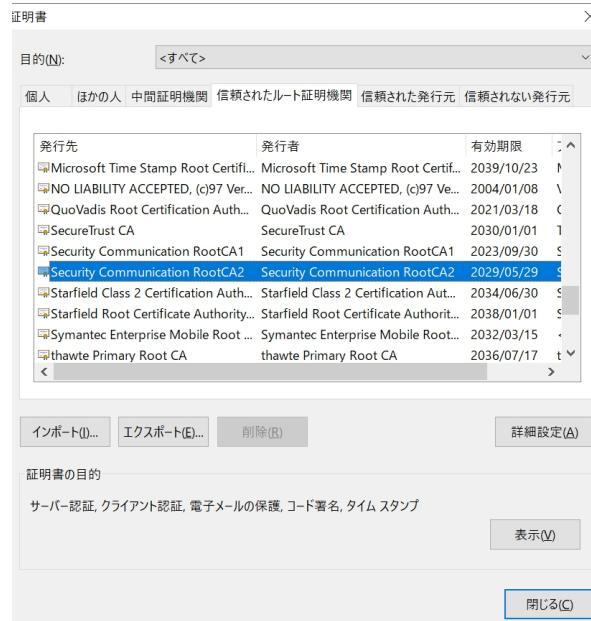
実は、中間 CA 証明書 2 を発行した D さんこと [Security Communication RootCA1] の正当性を証明する「ルート証明書」は、皆さんのが使っているブラウザに最初から入っています。Chrome で [設定] を開いたら [証明書] で検索（図 5.14）して、[証明書の管理] をクリックします。

5.7 SSL 証明書は異なる 3 つの仕事をしている



▲図 5.14 [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 5.15）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 5.15 [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。

5.7.6 <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^{*23}しました。SSL証明書には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、新しいSSL証明書に差し替えなければいけないのですが、この更新を怠って有効期限が切れてしまったことで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。たとえば株式会社はてなが提供している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超えるとアラートが発報^{*24}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。

5.8 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、DigiCert&Symantecは219,000円、GMOグローバルサインは59,800円、そして本著で買ったFujiSSLは1,000円でした。^{*25}

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？

5.8.1 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」という名前です。

^{*23} <https://twitter.com/MSFT365Status/status/1224351597624537088>

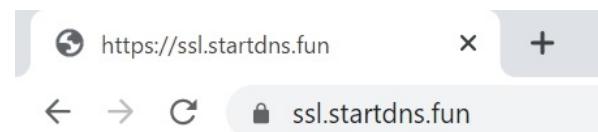
^{*24} URL外形監視にてSSL証明書の有効期限を監視できるようになりました - Mackerel ブログ <https://mackerel.io/ja/blog/entry/2016/01/29/115632>

^{*25} いずれも2020年2月時点、有効期間1年の税抜金額です

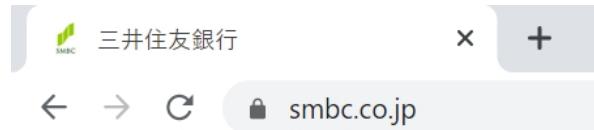
5.8 SSL 証明書はどうしてあんなに値段に差があるの？

- 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。
 - 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)

DV 証明書（図 5.16）と EV 証明書（図 5.17）のサイトを比較すると、値段も取得の煩雑さも EV 証明書の方が圧倒的に上ですが、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 5.16 本著で作ったサイトの DV 証明書の鍵マーク



▲図 5.17 三井住友銀行の EV 証明書の鍵マーク

5.8.2 さよならグリーンバー

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 5.18）され、一目で本物のサイトと判別ができました。



▲図 5.18 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 証明書だと運営組織名も表示（図 5.20）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。

5.8 SSL 証明書はどうしてあんなに値段に差があるの？



▲図 5.19 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 5.20 EV 証明書だと運営組織名も表示される

5.8.3 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が、元となるドメイン名や、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。

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

5.8.4 【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
- 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、example.comのSSL証明書も必要です。

5.8.5 <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

5.8 SSL 証明書はどうしてあんなに値段に差があるの？

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。

実は 1 つの IP アドレスで、複数の HTTPS のサイトを動かす名前ベースのバーチャルホストの場合、

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やりとりが行なわれるため、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、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、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^{*26}か、のどちらかです。

^{*26} 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

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.

そして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

本著（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/xxxxxx>
- パスワード : **xxxxxx**

Special Thanks:

- 私を仔猫だと思い込んでいるふさふさ茶色のやさしい猫に捧ぐ
- Gunnell Marshall

レビュアー

- Takeshi Matsuba
- Mari Kubota

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 制作会社のシステムエンジニア。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。モバイルサイトのエンジニア、SIer とソーシャルゲームの広報を経て、2013 年よりサーバホスティングサービスの構築と運用を担当したのち、再び Web アプリケーションエンジニアとしてシステム開発に従事。「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 4,5,6 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 7,800 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

2020-02-29/2020-03-01 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